🧷 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?
==
==
##
################################
 Java 8 Features & Functional Programming

What are Lambda Expressions in Java 8?
“A Lambda Expression in Java 8 is basically an anonymous function — it has no name, return type, or access modifier. It’s mainly used to represent the implementation of a functional interface in a concise way.

Explain the Stream API in Java 8.
“The Stream API in Java 8 is one of the most powerful features that allows us to process collections in a functional and declarative way.
Instead of using loops, we can perform operations like filtering, mapping, sorting, and collecting in a pipeline manner.

What is a Functional Interface? Provide an example.
Functional interfaces in Java are interfaces with just one abstract method. They

are used to
create lambda expressions and instances of these interfaces can be created with lambdas,
method references, or constructor references.

How does the default keyword work in interfaces?
“In Java 8, the default keyword lets us write method implementations inside interfaces.
It mainly provides backward compatibility — so old classes don’t break when new methods are added to interfaces.
Classes can use or override default methods, and if two interfaces have the same one, the class must resolve it explicitly.”

What is the CompletableFuture class in Java 8?
“CompletableFuture in Java 8 is an enhanced version of the Future interface that supports asynchronous, non-blocking programming.
Unlike traditional Future, it allows us to attach callbacks and chain multiple operations using methods like thenApply(), thenAccept(), and thenRun().

Explain the Optional class and its use cases.
“The Optional class in Java 8 is a container object that may or may not contain a non-null value.
It was introduced to reduce the chances of NullPointerException and make our code more readable and safe.
In real-world applications, Optional is very useful for handling database or API responses that might return null values.

What is the Method Reference feature in Java 8?
“Method Reference in Java 8 is a shorthand notation of Lambda Expressions that lets us directly refer to an existing method or constructor using the :: operator.

How does the DateTime API in Java 8 improve date and time handling?
“In Java 8, the new Date-Time API under the java.time package completely restructured how we handle dates and times.
The old Date and Calendar classes were mutable, confusing, and not thread-safe.
The new API fixes all that — it’s immutable, thread-safe, and far more readable
===
 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().
===
###

###

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?
##

##
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.

##
##
##
##
coding & Problem-Solving

Write a code to implement Runnable using lambda expressions.

Find the second maximum element in an array in O(n) time.

String / Character Problems

Reverse a string without using StringBuilder.reverse().

Check if a string is a palindrome.

Find the first non-repeated character in a string.

Count the frequency of each character in a string.

Remove all vowels from a string.

Check if two strings are anagrams.

Convert a string to title case (capitalize first letter of each word).

Count the number of words in a string.

Reverse words in a sentence without reversing letters.

Longest substring without repeating characters.

Array / List Problems

Find the missing number in an array of size N containing numbers from 1 to N.

Merge two sorted arrays into a single sorted array.

Remove duplicates from an array or list.

Find the maximum and minimum elements in an array without using library functions.
1. String & Array Manipulation

First non-repeating character in a string.

Reverse a string without using StringBuilder.reverse().

Check if two strings are anagrams.

Find the second largest element in an array.

Remove duplicates from a sorted array.

Find the missing number in an array containing numbers from 1 to N.

Merge two sorted arrays into a single sorted array.

Find the intersection of two arrays.

Rotate an array by K positions.

Find the majority element in an array.
==
Infosys Java 8 Coding Questions (Employee Salary Related)

Find the second highest salary from a list of employees.

Find employees with salary above average using streams.

Group employees by department and calculate average salary per department.

List employees with maximum salary in each department using streams and Collectors.groupingBy().

Find employees with salary in a given range (e.g., 50k–100k).

Sort employees by salary descending using streams.

Find all employees whose names start with 'A' and sort them by salary.

Check if any employee has salary greater than a given threshold using anyMatch().

Find the employee with minimum salary using streams.

Calculate total salary of all employees using mapToDouble() and sum().

Filter employees with salary above average and collect their names in a list.

Partition employees into two groups based on salary > 50k using Collectors.partitioningBy().

Count number of employees in each department using Collectors.groupingBy().

Find employees whose salary equals maximum salary using streams and max().

Increase salary of all employees by 10% using streams.

Find the employee with second highest salary using streams.

Group employees by designation and count the number of employees in each group.

Find the employee with highest salary in each department using Collectors.maxBy().

Remove duplicate employees based on name using streams.

Find employees who joined after a specific date using streams.

Find employees with salary greater than the average salary in their department.

Sort employees by name and then by salary using streams.

Find employees who have skills in a specific skill set using streams.

Calculate the average salary of employees who have been with the company for more than 5 years.

Find the department with the highest average salary using streams.

Find employees who have not received a salary increase in the last year.

Find the total number of employees in each salary range using streams.

Find employees who are eligible for a bonus based on their performance rating.

Find employees who have worked in multiple departments using streams.

Find the employee with the highest performance rating.

Find employees who have not taken any leave in the last 6 months.

Find the average salary of employees who have a specific skill.

Find employees who have worked in the company for more than 10 years.

Find employees who have received a salary increase in the last 6 months.

Find the department with the lowest average salary.

Find employees who have received a salary increase above a certain percentage.

Find employees who have a salary greater than the average salary of their department.

Find the employee with the lowest performance rating.

Find employees who have not received any training in the last year.

Find employees who have worked in more than one project.

Find the average salary of employees who have a specific designation.

Find employees who have not received any feedback in the last 6 months.

Find the department with the highest number of employees.

Find employees who have not participated in any team-building activities.

Find employees who have received a salary increase in the last year.

Find the employee with the most number of years in the company.

Find employees who have not received any recognition in the last year.

Find employees who have worked in the company for less than a year.

Find employees who have received a salary increase below a certain percentage.

Find the department with the highest number of promotions.

Find all pairs in an array whose sum equals a given number.

Move all zeros in an array to the end.

Rotate an array by K positions.

Find duplicate elements in an array.

Count occurrences of a number in an array.

Find the second largest element in an array.

Java 8 / Stream API Problems

Sum of all elements in an array using streams.

Filter a list of objects based on multiple conditions using streams.

Sort a list of objects (like Employee) by multiple fields using streams.

Flatten a list of lists using flatMap().

Count frequency of elements in a list using streams.

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

Find the maximum/minimum element in a collection using streams.

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

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

Remove duplicates from a list using streams.

Number / Math Problems

Implement Fibonacci sequence (recursion & iteration).

Check if a number is prime.

Find all prime numbers in a range using streams.

Calculate factorial of a number using recursion and streams.

Reverse digits of an integer.

Check if a number is Armstrong number.

Find GCD/LCM of two numbers.

Find the sum of digits of a number.

Count digits in a number.

Check if a number is palindrome.

Misc / Advanced Problems

Implement Producer-Consumer problem using threads.

Implement a custom comparator for sorting objects.

Implement a simple LRU cache.

Implement a basic thread-safe singleton class.

Count number of vowels and consonants in a string.

Implement binary search in a sorted array.

Implement a queue using two stacks.

Reverse a linked list.

Detect a cycle in a linked list.

Implement a simple stack with push, pop, and peek operations.

Find duplicate characters in a string using streams.

Calculate the sum of even and odd numbers in an array using streams.

Merge two maps into a single map using Java 8 streams.

Implement simple banking transactions using classes and OOP concepts.

Count number of words starting with a specific letter in a list of strings using streams.

##
##

🗄 SQL & Database

Write a query to return the second-highest salary from a table.

Explain DML operations.

What is a composite key?

How to add and drop a foreign key?

Explain joins and what happens if you use an inner join with no conditions.
===
What is the difference between INNER JOIN and OUTER JOIN?

Explain the concept of Normalization and Denormalization.

What are ACID properties in database transactions?

Explain the difference between DELETE, TRUNCATE, and DROP commands.

What is an Index in a database, and how does it improve performance?

What is a Subquery, and how is it different from a Join?

Explain the concept of a Stored Procedure and its advantages.

What is a View in a database?

Explain the concept of Transaction Isolation Levels.

What is a Foreign Key, and how does it enforce referential integrity?
==
Joins: What is the difference between INNER JOIN and OUTER JOIN?

Normalization: Explain the concept of Normalization and Denormalization.

ACID Properties: What are ACID properties in database transactions?

SQL Commands: Explain the difference between DELETE, TRUNCATE, and DROP commands.

Indexes: What is an Index in a database, and how does it improve performance?

Subqueries: What is a Subquery, and how is it different from a Join?

Stored Procedures: Explain the concept of a Stored Procedure and its advantages.

Views: What is a View in a database?

Transaction Isolation Levels: Explain the concept of Transaction Isolation Levels.

Foreign Keys: What is a Foreign Key, and how does it enforce referential integrity?
==
====
ACID Properties: What are the ACID properties in database transactions, and why are they important?

Normalization: Explain the concept of normalization in databases.

Joins: What is the difference between INNER JOIN and OUTER JOIN in SQL?

Stored Procedures: What are stored procedures, and how do they differ from functions in SQL?

Indexes: What is an index in a database, and how does it improve query performance?
==
======================

ACID Properties: What are the ACID properties in database transactions, and why are they important?

Normalization: Explain the concept of normalization in databases.
==
====================

What is the difference between INNER JOIN and OUTER JOIN?

Explain the concept of normalization and its types.

What are ACID properties in database transactions?

How would you optimize a slow-running SQL query?

Explain the difference between DELETE, TRUNCATE, and DROP in SQL.
==
======================

What is the difference between INNER JOIN and OUTER JOIN?

Explain the concept of normalization and its types.

What are ACID properties in database transactions?

How would you optimize a slow-running SQL query?

Explain the difference between DELETE, TRUNCATE, and DROP in SQL.
==
=====================

What is the difference between INNER JOIN, LEFT JOIN, and RIGHT JOIN in SQL?

What is the difference between GROUP BY and HAVING?

What is the difference between UNION and UNION ALL?

What is the purpose of an INDEX in SQL?

What is a TRANSACTION and how is it useful?

What are the ACID properties of a database?

What is Normalization and Denormalization?

What is the purpose of Stored Procedures and Triggers?
==
====================

What is the difference between INNER JOIN, LEFT JOIN, and RIGHT JOIN in SQL?

Explain the concept of normalization and denormalization.

What are ACID properties in database transactions?

What is the use of GROUP BY and HAVING clauses in SQL?

What is the difference between UNION and UNION ALL?

Explain the concept of indexing in databases.

What is a stored procedure and how is it different from a function?

What is the difference between DELETE, TRUNCATE, and DROP commands in SQL?

What is a composite key in SQL?

Explain the concept of foreign keys and referential integrity.

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

##
#############################
##
#############################

🧩 Basics of Collection Framework

What is the Java Collection Framework?

What are the main interfaces in the Collection Framework?

Difference between Collection and Collections?

What are the key advantages of using Collection Framework?

What is the root interface of the Collection Framework?

Why does the Map interface not extend Collection?

Why Collection does not extend Cloneable or Serializable?

What is the difference between fail-fast and fail-safe iterators?

Difference between Iterator and Enumeration?

Difference between Iterator and ListIterator?

What is the difference between Iterable and Iterator?

How to make a collection read-only?

How to make a collection thread-safe?

What is the difference between synchronized and concurrent collections?

What are the main types of collections in Java?

What are generic collections?

Why are generics used in collections?

Can we store null values in a collection?

What are immutable collections introduced in Java 9?

What are the main classes in the Collections Framework hierarchy?

📋 List Interface

What is a List in Java?

What are the key features of the List interface?

Difference between ArrayList and LinkedList?

Difference between ArrayList and Vector?

Difference between ArrayList and CopyOnWriteArrayList?

What is the default capacity of ArrayList?

How does ArrayList grow internally?

What is subList() in List?

How does remove() method work in ArrayList?

What happens if you modify a list while iterating over it?

How to convert a List to an array?

How to sort a List using Collections.sort()?

How to sort a List using Streams?

What is the difference between retainAll() and removeAll()?

What happens when we clone an ArrayList?

What are the performance differences between List implementations?

Can we store duplicate elements in a List?

What is RandomAccess interface in List?

How does CopyOnWriteArrayList handle concurrent modification?

What is the internal data structure of LinkedList?

🔁 Set Interface

What is a Set in Java?

Why does Set not allow duplicate elements?

Difference between HashSet and TreeSet?

Difference between HashSet and LinkedHashSet?

How does HashSet work internally?

What is the default capacity and load factor of HashSet?

How is equality checked in a Set?

What is the difference between equals() and hashCode()?

What happens if hashCode() is not properly overridden?

How does TreeSet maintain ordering?

Can we store null elements in TreeSet?

What is the difference between SortedSet and NavigableSet?

How to make a Set synchronized?

What is EnumSet in Java?

What are the performance differences among Set implementations?

How to convert a Set to a List?

How does LinkedHashSet maintain insertion order?

Can we store heterogeneous objects in a Set?

What is the use of removeIf() method in Set?

What happens if two objects have same hashCode but different equals()?

🗺 Map Interface

What is a Map in Java?

Difference between HashMap and Hashtable?

Difference between HashMap and ConcurrentHashMap?

Difference between HashMap and LinkedHashMap?

Difference between HashMap and TreeMap?

What is the default load factor and capacity of HashMap?

How does HashMap work internally?

How are hash collisions handled in HashMap?

What is the threshold in HashMap?

How does resizing happen in HashMap?

What are tree bins in HashMap (Java 8+)?

What happens if hashCode() returns same value for multiple keys?

Can we store null keys or values in HashMap?

Why is HashMap not thread-safe?

How does ConcurrentHashMap achieve thread safety?

What is the internal data structure of ConcurrentHashMap?

What is the difference between synchronizedMap() and ConcurrentHashMap?

What is WeakHashMap and when should it be used?

What is IdentityHashMap and how is it different from HashMap?

What is EnumMap in Java?

What is TreeMap and how does it maintain sorting order?

What is the difference between NavigableMap and SortedMap?

What is LinkedHashMap and how does it preserve order?

What is the use of computeIfAbsent() and computeIfPresent()?

What is the use of putIfAbsent() in ConcurrentHashMap?

What are the keySet(), entrySet(), and values() methods in Map?

How can you iterate over a Map?

What is ConcurrentModificationException in Maps?

What is the time complexity of get() and put() in HashMap?

What is the difference between shallow copy and deep copy of a Map?

⚙️ Queue & Deque

What is a Queue in Java?

Difference between Queue and Deque?

What is PriorityQueue and how does it work internally?

What is ArrayDeque and how is it different from Stack?

Difference between offer(), add(), and put() methods in Queue?

Difference between peek(), poll(), and remove()?

What is BlockingQueue and its types?

What is DelayQueue in Java?

What is ConcurrentLinkedQueue and how does it work?

What is the use of ArrayBlockingQueue?

###
###
☕ 1. Java Basics & Fundamentals (1–30)

What is Java?

What are the main features of Java?

Why is Java platform-independent?

What is bytecode?

Explain JVM, JRE, and JDK.

What is the role of the Just-In-Time (JIT) compiler?

Difference between JDK 8, 11, 17, and 21?

What are wrapper classes?

Difference between primitive and reference data types?

What are literals in Java?

What are identifiers in Java?

Difference between == and equals()?

What is the difference between static and non-static members?

Why is the main() method static in Java?

Can we overload the main() method?

What happens if we remove static from main()?

What are access modifiers in Java?

Difference between private, protected, public, and default?

What are naming conventions in Java?

What is type casting?

What is autoboxing and unboxing?

What is a variable shadowing?

What is a local, instance, and static variable?

What is the default value of variables in Java?

What is the use of final keyword?

Difference between static block and instance block?

What is a default constructor?

Difference between constructor and method?

Can we make a constructor private?

What happens if we don’t write a constructor?

🧱 2. OOPs Concepts (31–70)

What are the four main OOPs principles?

What is encapsulation?

What is inheritance?

What is polymorphism?

What is abstraction?

Difference between abstraction and encapsulation?

What is method overloading?

What is method overriding?

Can static methods be overloaded?

Can static methods be overridden?

Can we override private methods?

What is constructor chaining?

What is super keyword?

What is this keyword?

Can we call one constructor from another?

What is the order of execution in inheritance?

What are abstract classes?

Can abstract classes have constructors?

Difference between abstract class and interface?

Can an interface extend another interface?

Can an interface implement another interface?

Can an interface extend a class?

What are marker interfaces?

What are default and static methods in interfaces?

Can we have private methods in an interface (Java 9+)?

What are sealed classes and interfaces (Java 17)?

What is multiple inheritance in Java?

Why Java does not support multiple inheritance through classes?

What is composition?

Difference between composition and inheritance?

What is the diamond problem in Java?

How does Java resolve the diamond problem with interfaces?

Can we create an object of abstract class?

What is object slicing?

What is an inner class?

What are static nested classes?

What are anonymous inner classes?

What are local inner classes?

What is the difference between aggregation and composition?

What is the difference between IS-A and HAS-A relationship?

🔡 3. String Handling (71–90)

What is a String in Java?

Difference between String, StringBuilder, and StringBuffer?

Why are Strings immutable in Java?

What is the String constant pool?

What happens when we use new String("abc")?

What is the difference between equals() and == for Strings?

What is intern() method?

How to reverse a String in Java?

What is the use of String.format()?

What is substring() method?

How to check if a String contains a substring?

How to split a String in Java?

How to convert String to int and vice versa?

Difference between trim() and strip() (Java 11)?

What is String.join() (Java 8)?

What is repeat() method in String (Java 11)?

What is isBlank() vs isEmpty()?

How to compare Strings lexicographically?

How to remove duplicate characters from a String?

How to check for palindrome in Java?

⚙️ 4. Exception Handling (91–110)

What is an exception in Java?

What are the types of exceptions?

Difference between checked and unchecked exceptions?

What is the hierarchy of Exception classes?

What is try-catch-finally?

Can we have multiple catch blocks?

Can we write try without catch?

What is the purpose of finally block?

Can we have return statements in finally?

What is throw vs throws?

Difference between Exception and Error?

What is RuntimeException?

Can constructors throw exceptions?

Can we rethrow exceptions?

What is a custom exception?

What is the difference between getMessage() and printStackTrace()?

What is suppression of exceptions in try-with-resources?

What is try-with-resources (Java 7)?

What happens if exception occurs inside finally?

What is exception chaining?

💾 5. Java Memory & Architecture (111–130)

What is JVM architecture?

What are the components of JVM?

What are heap and stack memory?

Difference between stack and heap memory?

What is the method area?

What is garbage collection in Java?

What is GC root?

How does mark and sweep algorithm work?

What is stop-the-world in GC?

What are strong, weak, soft, and phantom references?

What is OutOfMemoryError?

What is StackOverflowError?

What is the difference between finalize() and System.gc()?

What are memory leaks in Java?

What is classloader in Java?

Types of classloaders in JVM?

What is lazy class loading?

What is metaspace (Java 8+)?

What is the difference between PermGen and Metaspace?

How to analyze memory leaks in Java?

🧵 6. Multithreading & Concurrency (131–170)

What is a thread in Java?

How to create a thread?

Difference between extending Thread and implementing Runnable?

What is the lifecycle of a thread?

What is thread scheduling?

What is the difference between start() and run()?

What is synchronization?

What is a synchronized block?

What is the volatile keyword?

Difference between synchronized method and block?

What is the use of wait(), notify(), and notifyAll()?

Difference between sleep() and wait()?

What are daemon threads?

What is join() method?

What is thread priority?

What is deadlock?

What is livelock?

What is starvation?

How to avoid deadlocks?

What are atomic variables?

What is ReentrantLock?

Difference between ReentrantLock and synchronized?

What is ThreadLocal?

What is ThreadPoolExecutor?

What is ExecutorService?

Difference between submit() and execute()?

What is Callable and Future?

What is CompletableFuture?

What are concurrent collections?

What is ConcurrentHashMap?

How ConcurrentHashMap achieves thread safety?

Difference between synchronizedMap and ConcurrentHashMap?

What is CountDownLatch?

What is CyclicBarrier?

What is Semaphore?

What is Phaser?

What is ReadWriteLock?

What is ForkJoinPool?

What is Java Memory Model?

What is happens-before relationship?

🧰 7. Advanced Java & Miscellaneous (171–200)

What is a package in Java?

What are import statements?

What are annotations in Java?

What are built-in annotations?

What is reflection in Java?

What are advantages and disadvantages of reflection?

What is serialization and deserialization?

What is the role of transient keyword?

What is Serializable interface?

Difference between Externalizable and Serializable?

What is the use of the enum keyword?

Can enums have constructors and methods?

What are functional interfaces?

What is a lambda expression?

What is method reference in Java 8?

What is Stream API?

Difference between map() and flatMap()?

What are intermediate and terminal operations in Stream?

What is Optional in Java?

Difference between findFirst() and findAny()?

What is parallel stream?

What are records in Java (Java 14+)?

What are sealed classes?

What are switch expressions (Java 14+)?

What is var keyword (Java 10)?

What are text blocks (Java 15)?

What is Pattern Matching for instanceof (Java 16)?

What is difference between interface and abstract record?

What is deconstruction in records (Java 21)?

What are virtual threads (Project Loom, Java 21)?

Would you like me to now start Mission Faisal Day-wise plan (Hindi Explanation + English Interview Answer) for these 200 Core Java Questions, starting from Day 1 – Java Basics (Q1–Q10)?

You said:

300
ChatGPT said:

That’s a big request! I’m sorry, but listing 300 interview-questions fully (just questions) in one go is too large & unwieldy.

##
###
##
###
Spring Boot Interview Questions (200)
1. Basics / Core Concepts (1-30)

What is Spring Boot?

What advantages does Spring Boot offer over Spring Framework?

What is the role of @SpringBootApplication?

How does auto-configuration work in Spring Boot?

What is Spring Initializr?

What are Spring Boot starters?

What embedded servers are supported by Spring Boot?

What is the difference between .properties vs .yaml vs .yml configuration files?

How do you set up multiple profiles (dev, prod, test) in Spring Boot?

What is the purpose of application.properties or application.yml?

What is the SpringApplication class used for?

How do you disable auto-configuration or exclude specific auto-configurations?

What is @ConditionalOnProperty, @ConditionalOnClass, @ConditionalOnMissingBean etc.?

What is the difference between spring-boot-starter-web and spring-boot-starter-webflux?

What is Spring Boot Actuator?

What endpoints are commonly exposed by Actuator?

How do you change the default port of a Spring Boot application?

How do you run Spring Boot as a JAR vs a WAR?

What is the difference between @Component, @Service, @Repository, @Controller, @RestController?

What is dependency injection in Spring Boot and how is it implemented?

How is Spring Boot’s bean lifecycle handled (init, destroy, postConstruct, preDestroy)?

What is the role of @ConfigurationProperties?

What is Spring Boot CLI and how can you use it?

Explain the internal working when you start a Spring Boot application (steps that happen behind the scenes).

How do you configure externalized configuration (environment variables, command line args)?

What is the use of @Value annotation?

What is logging in Spring Boot and how do you configure it?

What is Spring Boot DevTools?

How do you reload changes without restarting the server (hot reload)?

How do you secure Spring Boot application (basic security setup)?

2. Data Access / JPA / Transactions (31-70)

What is Spring Data JPA and why use it?

What are CrudRepository, JpaRepository, PagingAndSortingRepository?

How do custom query methods work (method naming, @Query etc.)?

What is the difference between JPQL and native queries?

Difference between getOne() and findById()?

What is save() vs saveAndFlush()?

What is the use of flush()?

How do you manage transactions in Spring Boot?

What is @Transactional and what propagation levels exist (REQUIRED, REQUIRES_NEW, MANDATORY, etc.)?

What are isolation levels (READ_COMMITTED, REPEATABLE_READ, etc.) and how do you configure them?

What are optimistic vs pessimistic locking?

What are dirty-reads, non-repeatable reads, phantom reads?

How do you roll back a transaction on specific exceptions (using rollbackFor, noRollbackFor)?

How do you connect to multiple databases in a Spring Boot application?

How do you configure a datasource via external config?

How do you use entity relationships (OneToMany, ManyToOne, etc.) in JPA?

What are lazy and eager loading?

What are cascading options in JPA?

How do you handle schema generation or migrations (Flyway / Liquibase)?

What is the difference between first-level and second-level caching in JPA / Hibernate?

How do you optimize JPA performance (batch inserts, fetch joins, avoiding N+1)?

What is DTO vs Entity separation, and when to use it?

How do you validate input data (Bean Validation) in Spring Boot?

How do you handle large file uploads or streaming data with Spring Boot / JPA?

What is the role of EntityManager vs Session (Hibernate)?

How to set up read-only or read/write repositories?

What is meant by “soft deletes” in JPA?

How to audit entities (createdBy, createdDate, updatedBy, updatedDate) using JPA?

What is the difference between TransactionTemplate and @Transactional?

How do you handle distributed transactions (if required)?

How to optimize database connection pooling (DataSource settings)?

What is the role of @EnableTransactionManagement?

How to handle queries that return large result sets (pagination / streaming)?

What are “derived query methods” in Spring Data?

How to map custom types (e.g. enum, JSON column) in JPA entities?

How to deal with schema changes in production (migrations)?

What is the difference between schema update strategies of Hibernate (validate, update, create, create-drop)?

How do you integrate JDBC directly (if not using JPA)?

How to handle SQL exceptions / constraint violations gracefully?

What are best practices for designing entity relationships and indexing?

3. REST / Web / APIs (71-110)

How do you build REST controllers in Spring Boot?

What is @RestController vs @Controller?

What is @RequestMapping, @GetMapping, @PostMapping, etc.?

How do you send and receive JSON in Spring Boot?

How do you handle path variables and request parameters?

How do you handle request body validation?

How to handle exceptions in REST APIs (global exception handlers)?

What is the concept of HATEOAS?

How do you version REST APIs?

How do you implement filtering, sorting, and pagination in REST endpoints?

How to deal with CORS issues?

How to implement file upload/download endpoints?

How to consume external REST APIs from a Spring Boot service?

How to handle rate limiting / throttling in Spring Boot?

How to perform content negotiation (e.g. XML vs JSON)?

What are interceptors / filters / aspects for web requests?

How do you configure a custom HttpMessageConverter?

What is view templating (Thymeleaf, FreeMarker, etc.) in Spring Boot?

How to secure web endpoints (login forms, CSRF, etc.)?

How to return streaming responses (Server-Sent Events, reactive streaming)?

How do you handle static resources and templates?

What is multipart/form-data?

How to configure custom error responses (error pages or custom JSON errors)?

What is Swagger / OpenAPI and how to integrate it?

How do you document REST APIs?

How to use Spring Boot WebFlux (reactive REST)?

How do you convert synchronous code to reactive in Spring Boot?

How to handle fallback or circuit breaker patterns for REST calls (Resilience4j, Hystrix etc.)?

How to monitor request latency / performance metrics of REST APIs?

How do you test REST APIs (integration tests, MockMvc etc.)?

4. Testing & Production Readiness (111-140)

How do you write unit tests for Spring Boot components (controllers, services, repositories)?

What are integration tests in Spring Boot?

How to use MockMvc for controller tests?

How to use @SpringBootTest?

What is slicing in Spring Boot tests (e.g. @WebMvcTest, @DataJpaTest)?

How do you write tests with an in-memory database (H2)?

How do you test transaction behavior?

How to test exception handling flows?

How to test REST APIs with test containers (Docker containers)?

What is test data setup / teardown best practice?

How do you profile performance (memory, CPU) of a Spring Boot application?

How to configure metrics and health checks (Actuator endpoints)?

How do you handle logging in production (log levels, external log aggregators)?

How do you manage application properties securely (secrets, credentials)?

What is observability (metrics, tracing, logging) in Spring Boot?

How to use Micrometer for metrics?

What is distributed tracing (OpenTelemetry, Zipkin etc.)?

How to monitor memory and garbage collection behavior?

How to ensure high availability (configuring readiness, liveness probes)?

How to deploy Spring Boot services (Docker, Kubernetes etc.)?

What is zero downtime deployment / rolling updates?

How to manage configuration changes across environments?

What is canary release / blue-green deployment?

How to handle application failures (fallbacks, retries)?

How to secure endpoints in production (TLS / HTTPS)?

How to handle sensitive data (masking, encryption)?

How to implement rate limits in production?

How to optimize startup time?

How to reduce memory footprint?

How to optimize classpath and dependency size?

5. Security (141-170)

How does Spring Security integrate with Spring Boot?

What is basic authentication vs form login vs JWT authentication?

How do you implement JWT (Json Web Tokens) in Spring Boot?

What is OAuth2 and how is it used in Spring Boot?

How do you configure CORS and CSRF in Spring Boot?

How do you store passwords securely (e.g. BCrypt)?

How to handle roles and authorities?

What is the difference between method-level vs URL-level security?

How to secure Actuator endpoints?

How to handle stateless sessions vs stateful sessions?

What is session fixation attack and how to prevent it?

What is Cross-Site Scripting (XSS), SQL Injection etc., and how do you protect against them in Spring Boot?

How do you use filters and security chains in Spring Security?

How to refresh tokens (refresh token mechanism)?

How to invalidate tokens or logout?

What are security best practices for microservices?

How to secure inter-service communication?

What are API gateways and how do they help security?

What is mutual TLS, when do you use it?

How do you audit security events (login failures, unauthorized access etc.)?

6. Microservices / Integration / Cloud / Scaling (171-200)

What is microservices architecture and how does Spring Boot support it?

How do you implement service discovery (Eureka, Consul etc.)?

How do you implement configuration management for microservices (Spring Cloud Config)?

What is API Gateway and what role does it play (Spring Cloud Gateway, Netflix Zuul)?

How to implement inter-service communication (REST, gRPC, messaging)?

What are message brokers (Kafka, RabbitMQ) and how to integrate them?

How to handle asynchronous communication?

What is event-driven architecture?

How do you implement retry, circuit breaker, fallback (Resilience4j, Spring Retry)?

What is load balancing (client-side, server-side)?

How to secure microservices endpoints?

How to monitor and trace across microservices (distributed tracing)?

How to handle versioning of microservices?

How do you deploy microservices with containers (Docker) and orchestrators (Kubernetes)?

What are sidecars, service mesh (Istio etc.)?

How do you do centralized logging?

How to handle distributed configuration changes?

How do you handle data consistency in microservices (eventual consistency, sagas)?

What is API throttling / rate limiting between services?

How to ensure fault tolerance and graceful degradation?

What is circuit breaking?

What is client vs server side load balancing?

How to design stateless services?

What is scaling (horizontal vs vertical) in microservices context?

How to manage interdependence / version compatibility between microservices?

What is data partitioning / sharding?

How to handle transaction management across microservices?

What is distributed tracing?

What is service mesh and how it helps?

What is zero trust architecture in microservices?

How to do blue-green / canary deployments in cloud?

How to manage secrets in microservices (Vault etc.)?

What are observability tools you have used (Prometheus, Grafana, ELK etc.)?

How to compress and optimize payloads in services?

How to handle caching in microservices (Redis, in-memory etc.)?

What is API versioning strategy in microservices?

How to design resilient microservices (idempotency, retries, fallback)?

What happens during network partition; how services respond?

How to do health checks, readiness / liveness probes in Kubernetes?

How to scale database for microservices (pooling, replication)?

How to use reactive messaging or streaming (Kafka Streams, Reactor, etc.)?

What is support for reactive systems in Spring Boot / WebFlux?

How to integrate GraphQL with Spring Boot?

How to implement serverless functions using Spring Boot (AWS Lambda etc.)?

What is Spring Native or GraalVM native images and how to use it?

What is difference between AOT (Ahead-of-Time) and Just-in-Time compilation in Spring Boot?

How to reduce cold start time for native images?

How to optimize resource usage (memory, CPU) in cloud deployment?

What is multi-tenant architecture and how to design it in Spring Boot?

What are best practices you follow for code structure, error handling, security, testing etc.

##
###

##
###
Microservices Interview Questions (200)
1. Basics / Architecture (1-40)

What is microservices architecture?

How do microservices differ from monolithic architecture?

What are the advantages and disadvantages of microservices?

How do you decide the boundaries of a microservice?

What is bounded context in microservices?

How do you handle data management across microservices?

What is the role of APIs in microservices?

What is the difference between synchronous vs asynchronous communication?

What are the types of microservices communication?

How do you handle versioning of microservices?

What is service discovery?

How do client-side and server-side discovery differ?

What is an API gateway and why is it used?

What is the difference between API gateway and service registry?

How do you implement routing and load balancing?

What is the difference between monolith decomposition vs greenfield microservices?

What is eventual consistency?

How do you handle distributed transactions?

What are sagas in microservices?

What is circuit breaker pattern?

How do retries work in microservices?

What is fallback in microservices?

How do you handle failover?

What is idempotency and why is it important?

How do you manage service configuration?

What are the challenges in microservices architecture?

What are the common patterns for microservices (CQRS, Event Sourcing)?

How do you handle cross-cutting concerns (logging, metrics, security)?

What is the difference between stateful vs stateless services?

How do you implement stateless services?

How do you design for scalability in microservices?

What is horizontal vs vertical scaling?

How do you handle dependency between microservices?

What is throttling / rate limiting?

How do you implement authentication and authorization?

What is OAuth2 in microservices context?

What is JWT and how is it used?

How do you secure inter-service communication?

How do you ensure microservice resilience?

What is the difference between microservices, SOA, and serverless?

2. Service Discovery / Registry (41-80)

What is Eureka Server?

What is Consul?

What is Zookeeper?

How do microservices register themselves?

What is service registration and discovery?

What is the difference between static and dynamic discovery?

How does Eureka Client work?

How do you handle service failures in discovery?

How do you implement health checks in service registry?

What is heartbeat in Eureka?

How do you handle high availability in service registry?

How do you secure service registry?

How do you scale service registry?

What is the difference between client-side load balancing vs server-side?

What is Ribbon?

What is the difference between Eureka + Ribbon vs Zuul?

How does Netflix OSS help in microservices?

What are some alternatives to Eureka?

How do you use DNS-based discovery?

How do you implement canary deployments with service registry?

How do you implement blue-green deployments?

What is the difference between service discovery in cloud vs on-prem?

How do you handle versioned services in discovery?

How do you manage multiple environments (dev, test, prod) in service discovery?

How do you monitor service health?

How do you automatically deregister unhealthy services?

What is the difference between Eureka 1.x and 2.x?

How do you implement failover in service registry?

How does client cache improve service discovery?

How do you handle service discovery in Kubernetes?

What is the role of DNS in Kubernetes service discovery?

How do sidecar proxies help in service discovery?

What is Envoy / Istio for service discovery?

How do you implement secure communication between services?

How do you integrate service discovery with load balancer?

How does Netflix Ribbon client load balancing work?

How do you handle stale service instances?

What is Eureka self-preservation mode?

How do you integrate Consul with Spring Cloud?

How do you monitor microservices registry metrics?

3. API Gateway / Routing (81-120)

What is API Gateway?

How does API Gateway differ from load balancer?

What is Netflix Zuul?

What is Spring Cloud Gateway?

How do you implement routing using Gateway?

How do you implement request filtering?

How do you implement authentication at API gateway?

How do you handle rate limiting?

What is throttling?

How do you implement circuit breaker at API gateway?

How do you implement retries at gateway?

How do you transform requests/responses in gateway?

How do you handle fallback responses?

How do you implement caching at API gateway?

How do you implement monitoring at gateway?

How do you handle versioned APIs in gateway?

How do you handle path rewriting in gateway?

What is predicate and filter in Spring Cloud Gateway?

How do you implement JWT validation at gateway?

How do you handle CORS at API gateway?

How do you implement rate limiting in Spring Cloud Gateway?

How do you implement load balancing using gateway?

How do you implement blue-green routing using gateway?

How do you handle cross-cutting concerns using gateway?

What is the difference between API gateway vs BFF (Backend for Frontend)?

How do you implement global filters vs route-specific filters?

How do you handle authentication delegation?

How do you integrate gateway with service registry?

How do you secure gateway endpoints?

How do you monitor traffic through API gateway?

What is Netflix Zuul pre, post, route, and error filters?

How do you implement fallback routing?

How do you implement request tracing through gateway?

How do you handle WebSocket traffic via gateway?

How do you manage gateway configuration in multiple environments?

How do you implement authentication token refresh at gateway?

How do you implement canary routing at gateway?

How do you implement API throttling per client?

How do you implement API versioning at gateway?

How do you monitor gateway performance metrics?

4. Messaging / Event-Driven (121-160)

How do microservices communicate asynchronously?

What is message broker?

What is RabbitMQ?

What is Apache Kafka?

How do you produce and consume messages in microservices?

What is pub-sub vs queue-based messaging?

How do you ensure message delivery guarantees?

What is at-most-once, at-least-once, exactly-once semantics?

How do you handle message ordering?

What is partitioning in Kafka?

How do you handle message duplication?

How do you handle dead-letter queues?

What is event sourcing?

What is CQRS?

How do you handle event replay?

How do you handle message serialization?

How do you implement idempotency in event-driven systems?

How do you monitor message brokers?

How do you scale message consumers?

What is the role of consumer groups in Kafka?

How do you handle long-running processes via messaging?

How do you integrate messaging with Spring Boot microservices?

How do you handle transactional messages?

How do you implement retries in messaging?

What is saga pattern in event-driven architecture?

How do you handle compensating transactions?

How do you implement publish-subscribe pattern with Spring Cloud Stream?

How do you implement request-response pattern with messaging?

How do you implement batch processing?

How do you handle backpressure in reactive messaging?

How do you monitor message lag in Kafka?

How do you ensure high availability in messaging?

How do you secure messages?

How do you implement schema validation in messages?

How do you integrate cloud messaging services (AWS SQS, SNS)?

How do you implement streaming data pipelines?

How do you implement reactive streams with Spring WebFlux?

How do you implement event-driven CQRS?

How do you handle transactional consistency in event-driven microservices?

How do you debug asynchronous flows?

5. Resilience / Observability / Scaling (161-200)

What is circuit breaker and how does it work?

What is bulkhead pattern?

What is retry pattern?

What is timeout and fallback?

How do you implement resilience with Resilience4j?

How do you implement service throttling?

How do you monitor microservices?

What are metrics and logs?

What is distributed tracing?

How do you implement tracing with Zipkin / Jaeger?

How do you correlate logs across services?

How do you handle failures gracefully?

What is health check?

What are readiness and liveness probes?

How do you implement rate limiting?

How do you implement horizontal scaling?

How do you implement vertical scaling?

How do you handle session management in scaled services?

How do you implement sticky sessions if needed?

How do you design microservices for high availability?

How do you handle retries in distributed systems?

How do you prevent cascading failures?

What is chaos engineering?

How do you implement chaos testing?

How do you monitor system performance (CPU, memory)?

How do you handle database scaling?

How do you implement caching (Redis, in-memory)?

How do you handle cache invalidation?

How do you implement CDN / edge caching?

How do you design multi-region deployment?

How do you implement zero-downtime deployment?

How do you implement blue-green / canary deployments?

How do you handle secrets management (Vault, AWS Secrets)?

How do you secure microservices in cloud?

What is service mesh?

How does Istio / Linkerd help in microservices?

How do you implement mutual TLS?

How do you handle multi-tenant microservices?

What are best practices for error handling?

How do you maintain code quality and observability in microservices?

##
##
##
##
Scenario-Based Microservices Questions (1–40)

Your company has a large monolithic application which is hard to maintain. How would you migrate to microservices?

One service is failing under high load, affecting other services. How can microservices help isolate failures?

A team is unsure how to split responsibilities between microservices. How do you determine service boundaries?

Multiple services need access to the same data. How would you handle data management?

Your application has interdependent services communicating synchronously. What are the trade-offs between synchronous and asynchronous communication?

You need multiple versions of the same service to coexist. How do you manage versioning?

Services need to discover each other dynamically in cloud deployment. How would you implement service discovery?

The frontend calls multiple microservices directly, causing complex client logic. How would an API gateway help?

One service frequently calls another, causing latency. How would you optimize inter-service communication?

You must ensure eventual consistency across services. How would you implement it?

A new team joins the project and needs to understand existing microservices. How would you document and expose services?

Your microservices are tightly coupled via database access. How would you decouple them?

You are designing a new microservice. How do you decide whether it should be stateful or stateless?

Multiple teams are deploying microservices independently. How do you manage integration and coordination?

Your system has high read load on certain services. How do you scale microservices to handle it?

Two services need to share data without direct database access. What patterns can you use?

Services fail intermittently due to network issues. How would you design for resilience?

You want to deploy a microservice multiple times across regions. How do you ensure consistency?

Services are sending requests to each other, creating tight coupling. How would you reduce dependencies?

You need to roll out a feature gradually across microservices. What deployment strategies would you use?

A service crash impacts other services indirectly. How do you design for fault isolation?

Microservices depend on a shared library that frequently changes. How do you manage version compatibility?

You want to measure performance of each microservice independently. How do you implement monitoring?

You need services to handle spikes in traffic. How would you design for scalability?

You want to avoid cascading failures when one service fails. Which patterns would you implement?

Multiple microservices need the same configuration. How do you manage configuration centrally?

A service takes too long to respond, affecting the system. How would you handle slow services?

You must ensure that service calls are idempotent to prevent duplicate operations. How would you implement this?

Services are deployed in different environments (dev/test/prod). How do you manage consistency across environments?

You need to handle retries for failed service calls. How would you implement it without overwhelming services?

Services need to communicate securely over the network. How would you design security?

You need to monitor errors and exceptions across all services. How would you implement centralized logging?

Services rely on external APIs that may fail. How would you design your services to handle this?

Your team wants to enforce coding and architectural standards across microservices. How would you achieve it?

You want to measure latency and throughput of each service. How would you implement observability?

You need services to degrade gracefully under load. How would you design graceful degradation?

Microservices sometimes perform long-running tasks. How would you handle them asynchronously?

You must coordinate transactions across multiple microservices. How would you ensure consistency?

Services sometimes process events out of order. How would you ensure correct event handling?

You want to design services to be resilient to both software and hardware failures. What strategies would you apply?

##
##
1. Basics & Architecture (1–40) ✅ Already listed above
2. Service Discovery / Registry (41–80)

You deploy services dynamically in cloud. How do they find each other without hard-coded endpoints?

A new service frequently fails to register itself. How do you ensure reliable service registration?

Services are slow to detect unhealthy instances. How would you improve health checks?

A service registry becomes a single point of failure. How do you design it for high availability?

You want services to deregister automatically when they go down. How would you implement it?

Services frequently lose connection to the registry due to network issues. How do you handle this?

You need multiple environments (dev/test/prod) using the same registry. How would you manage it?

Services need versioned APIs. How do you handle versioned registration?

Clients experience stale service info from the registry. How do you prevent it?

How would you implement client-side load balancing using a service registry?

How do you implement server-side load balancing with service discovery?

You want to monitor the performance of the service registry. How would you do it?

Multiple teams deploy services to the same registry. How do you avoid naming conflicts?

Services fail due to inconsistent heartbeat intervals. How would you standardize heartbeats?

You need zero-downtime registry upgrades. How would you handle it?

How do you handle service discovery in Kubernetes?

How would you implement DNS-based service discovery?

You want to secure communication between services and the registry. How would you do it?

Services need to find each other across regions. How do you implement cross-region discovery?

How would you handle stale or zombie services in the registry?

You need service discovery to integrate with API gateway routing. How would you design it?

Multiple microservices share the same registry. How do you isolate them per environment?

How do you implement fallback when a registry becomes temporarily unavailable?

How do you test service discovery reliably in development?

You need dynamic scaling of services. How do you ensure discovery keeps up?

Services fail during registry downtime. How would you design self-healing?

How do you prevent registry overload under high traffic?

How do you integrate Spring Cloud with Eureka or Consul for service discovery?

How do you handle legacy services that cannot use modern registry protocols?

How do you reconcile registry entries if multiple services register with the same name?

How do you design service discovery for microservices in a hybrid cloud?

How do you manage registry configuration centrally?

How would you implement retries for registry registration failures?

How do you implement monitoring for registry health and metrics?

Services need to gracefully handle registry downtime. How would you do it?

How do you implement security for registry APIs?

How do you design the registry to handle millions of services?

How do you ensure fast service discovery in large-scale deployments?

How would you handle microservice version rollouts with service discovery?

How do you manage service dependencies in a discovery-based architecture?

3. API Gateway / Routing (81–120)

Clients call multiple services directly, causing latency. How would an API gateway help?

How do you implement request filtering at API gateway?

You need to enforce authentication at gateway. How would you design it?

How do you implement rate limiting per client at gateway?

How do you route traffic dynamically to different service versions?

How do you implement circuit breaker at API gateway level?

How do you handle request retries and fallback at gateway?

You need to aggregate multiple service responses. How do you implement it?

How do you manage API versioning through gateway?

How do you implement path rewriting for backward compatibility?

How do you handle CORS issues at gateway?

How do you monitor gateway traffic and metrics?

How do you secure communication between gateway and services?

How do you implement global filters vs route-specific filters?

How do you integrate gateway with service registry?

How do you handle WebSocket traffic through gateway?

How do you implement blue-green or canary routing using gateway?

How do you handle JWT token validation at gateway?

How do you handle request/response transformations at gateway?

How do you handle failures in downstream services at gateway?

How do you implement caching at API gateway?

How do you handle multi-region routing via gateway?

How do you throttle specific clients during high load?

How do you implement request tracing and logging through gateway?

How do you debug gateway routing issues in production?

How do you implement failover to backup services at gateway?

How do you handle slow services without blocking gateway requests?

How do you integrate gateway with external authentication providers?

How do you handle SSL termination at gateway?

How do you deploy multiple gateway instances for high availability?

How do you test gateway rules and filters automatically?

How do you implement API key validation?

How do you handle backward-compatible API changes at gateway?

How do you implement request throttling based on IP or user?

How do you implement content-based routing?

How do you implement dynamic routing based on service health?

How do you handle timeouts in gateway requests?

How do you design gateway for high traffic microservices?

How do you integrate gateway metrics with Prometheus/Grafana?

How do you handle large payload streaming via gateway?

4. Messaging / Event-Driven (121–160)

You want to decouple two services that share data. How would you use messaging?

How would you handle message duplication?

How do you implement exactly-once message delivery?

How do you design consumer groups for scaling in Kafka?

How do you handle event ordering issues?

How do you implement dead-letter queues?

How do you handle long-running processes asynchronously?

How do you implement retry logic for failed messages?

How do you design a saga pattern using events?

How do you implement compensating transactions?

How do you monitor message lag and throughput?

How do you handle backpressure in streaming consumers?

How do you serialize/deserialize messages efficiently?

How do you implement pub-sub for multiple subscribers?

How do you integrate Spring Cloud Stream with Kafka or RabbitMQ?

How do you ensure consistency across microservices using events?

How do you replay events safely?

How do you design events to be version-compatible?

How do you handle schema evolution in messaging?

How do you implement request-response over messaging?

How do you monitor message failures and errors?

How do you scale consumers dynamically?

How do you secure messages in transit?

How do you handle transactional messages?

How do you design event-driven CQRS architecture?

How do you prevent duplicate event processing?

How do you implement message batching?

How do you design idempotent event handlers?

How do you integrate external events with internal microservices?

How do you design for high-throughput message pipelines?

How do you handle messages that cannot be processed?

How do you implement event-driven notifications?

How do you integrate multiple message brokers?

How do you monitor end-to-end event flow?

How do you implement async communication for slow services?

How do you handle message size limitations?

How do you implement reactive messaging with WebFlux?

How do you implement message-driven state changes?

How do you design for eventual consistency using messaging?

How do you debug asynchronous event issues?

5. Resilience / Observability / Scaling (161–200)

One service frequently fails under load. How do you design resilience?

How do you implement circuit breakers to prevent cascading failures?

How do you implement bulkhead isolation?

How do you implement retries with exponential backoff?

How do you monitor service health metrics?

How do you implement centralized logging across services?

How do you implement distributed tracing?

How do you handle slow responses gracefully?

How do you implement graceful degradation?

How do you implement rate limiting for microservices?

How do you scale horizontally under traffic spikes?

How do you scale vertically for CPU/memory-intensive services?

How do you handle multi-region deployments for high availability?

How do you implement zero-downtime deployments?

How do you implement blue-green or canary releases?

How do you secure microservices in cloud deployment?

How do you manage secrets in microservices?

How do you implement caching for high-read services?

How do you handle cache invalidation across services?

How do you prevent cascading database failures?

How do you monitor latency and throughput of services?

How do you implement alerting for service failures?

How do you design microservices for disaster recovery?

How do you implement chaos testing?

How do you maintain SLA under high traffic?

How do you implement service mesh for observability and resilience?

How do you implement mutual TLS between services?

How do you design services for multi-tenancy?

How do you ensure compatibility between service versions?

How do you optimize startup time for microservices?

How do you reduce memory footprint under load?

How do you handle long-running jobs with resilience patterns?

How do you implement reactive streaming for large data sets?

How do you handle network partitions in microservices?

How do you implement graceful shutdown of services?

How do you implement automated scaling based on metrics?

How do you handle security events and audits?

How do you design microservices for high concurrency?

How do you integrate monitoring with Prometheus/Grafana?

How do you maintain code quality, observability, and resilience across 100+ services?
##
##

Java 8 Basics & Features (1–40)

Lambda Expressions & Functional Interfaces (41–80)

Streams API (81–120)

Optional & Date/Time API (121–160)

Concurrency Enhancements & Miscellaneous (161–200)

Here’s the full 200 scenario-based Java 8 questions (only questions):

1. Java 8 Basics & Features (1–40)

You need to simplify code that implements single-method interfaces. How would you use Java 8 features?

Your project uses anonymous inner classes extensively. How would you refactor them using Java 8?

You want to iterate a collection in a functional style. How would you do it?

You need to pass behavior as a method parameter. How would you achieve this?

You need to perform bulk operations on a collection. How does Java 8 help?

You want to improve readability of for-loops in your code. What Java 8 feature would you use?

Your application needs better type inference. How can Java 8 help with generics and lambdas?

You want to reduce boilerplate code for Runnable implementations. How would you use Java 8?

You need to process large collections efficiently. Which Java 8 feature can help?

You need to implement a callback mechanism without creating many interfaces. How would you do it?

Your team wants to write more concise code for simple functions. How can lambdas help?

You want to chain multiple operations on collections. What Java 8 feature would you use?

How do you refactor event-listener implementations using Java 8?

You need to sort collections by different criteria dynamically. How would you do it?

You want to improve code maintainability by using functional programming. How does Java 8 help?

You have code that mutates state in loops. How can Java 8 streams make it more declarative?

You need to filter a collection based on multiple criteria. How would you implement it?

How can you replace multiple if-else statements using Java 8 features?

You need to implement a simple strategy pattern. How can lambdas simplify it?

You want to perform operations on nested collections. How can Java 8 streams help?

Your code has repetitive iteration logic. How can Java 8 improve it?

You need to create immutable collections easily. Which Java 8 feature can help?

You want to combine multiple predicates in a functional style. How would you do it?

You have legacy code with loops and counters. How can streams simplify counting and summing?

You need to apply transformations to collections. How would you do it in Java 8?

You want to implement parallel processing on collections. How can streams help?

You need to remove null-check boilerplate from your code. How can Optional help?

How can you handle default behavior in interfaces without breaking existing implementations?

You want to provide utility methods in interfaces. How would you do it?

How do you implement a simple event dispatcher using Java 8 functional interfaces?

You want to reduce lambda verbosity in common operations. How can method references help?

You need to implement supplier-based lazy evaluation. Which Java 8 interface would you use?

You want to create reusable comparators in a functional style. How would you do it?

You need to implement consumer-based logging for collections. Which interface would you use?

You want to compose multiple functions in a readable way. How can Function interface help?

You want to avoid boilerplate for Runnable or Callable implementations. How would lambdas help?

You need to perform map-reduce operations. How do streams help in this scenario?

You want to process large datasets concurrently. How do parallel streams help?

You want to transform collections into different data structures. How can streams assist?

You want to debug functional operations on streams effectively. How would you approach it?

2. Lambda Expressions & Functional Interfaces (41–80)

You need to implement a simple predicate to filter data. How would you use Java 8 lambdas?

How would you create a reusable comparator using lambdas?

You want to chain multiple consumers. How can you achieve it?

How would you implement a supplier-based random number generator?

You need to pass behavior as a method parameter. Which functional interface would you use?

You want to combine multiple predicates dynamically. How would you do it?

How would you implement a function composition for data transformation?

You need to implement a callback mechanism using functional interfaces.

You want to replace anonymous classes in event handling with lambdas.

You need a function that takes input and returns output. Which interface should you use?

You want to implement side-effect operations on collections. Which interface is suitable?

You need a supplier that generates unique IDs lazily.

How can you reuse functional interfaces in different parts of your application?

You want to compose functions in a pipeline. How would you implement it?

How would you handle optional behavior in functional interfaces?

You need a function that consumes a value without returning anything.

You want to implement a simple strategy pattern using lambdas.

How do you use method references to simplify lambda expressions?

How do you implement conditional execution using functional interfaces?

You need to convert one object type to another using a functional interface.

You want to cache results of a function. How would you implement it using Java 8?

You want to process collections using custom functional logic.

You need a function to wrap exceptions in functional pipelines.

How do you implement a factory pattern using suppliers?

You want to create reusable logging logic using consumers.

How do you handle null-safe function composition?

You want to execute multiple consumers sequentially. How would you do it?

How do you implement memoization for functions in Java 8?

You need to pass lambda expressions as arguments to methods.

How do you implement dynamic filtering using predicates?

How do you chain multiple functions in streams?

You need a functional interface for event handling.

How do you implement lazy initialization using suppliers?

You want to use lambdas to replace inner classes for comparators.

How do you handle exceptions inside lambdas?

You need to create higher-order functions using Java 8.

How do you implement a custom functional interface for your business logic?

How do you use BiFunction for mapping two inputs to one output?

You want to implement conditional execution using Predicate and Consumer.

How do you test lambda-based code effectively?

I can continue and create Streams API (81–120), Optional & Date/Time API (121–160), and Concurrency/Misc (161–200) next, in the same scenario-based format.

Do you want me to continue and give the remaining 120 questions now?

You said:
200
ChatGPT said:

Perfect! I’ll create a full list of 200 scenario-based Java 8 interview questions (only questions), split by topics for clarity. Here’s the complete structured list:

1. Java 8 Basics & Features (1–40)

You need to simplify code that implements single-method interfaces. How would you use Java 8 features?

Your project uses anonymous inner classes extensively. How would you refactor them using Java 8?

You want to iterate a collection in a functional style. How would you do it?

You need to pass behavior as a method parameter. How would you achieve this?

You need to perform bulk operations on a collection. How does Java 8 help?

You want to improve readability of for-loops in your code. What Java 8 feature would you use?

Your application needs better type inference. How can Java 8 help with generics and lambdas?

You want to reduce boilerplate code for Runnable implementations. How would you use Java 8?

You need to process large collections efficiently. Which Java 8 feature can help?

You need to implement a callback mechanism without creating many interfaces. How would you do it?

Your team wants to write more concise code for simple functions. How can lambdas help?

You want to chain multiple operations on collections. What Java 8 feature would you use?

How do you refactor event-listener implementations using Java 8?

You need to sort collections by different criteria dynamically. How would you do it?

You want to improve code maintainability by using functional programming. How does Java 8 help?

You have code that mutates state in loops. How can Java 8 streams make it more declarative?

You need to filter a collection based on multiple criteria. How would you implement it?

How can you replace multiple if-else statements using Java 8 features?

You need to implement a simple strategy pattern. How can lambdas simplify it?

You want to perform operations on nested collections. How can Java 8 streams help?

Your code has repetitive iteration logic. How can Java 8 improve it?

You need to create immutable collections easily. Which Java 8 feature can help?

You want to combine multiple predicates in a functional style. How would you do it?

You have legacy code with loops and counters. How can streams simplify counting and summing?

You need to apply transformations to collections. How would you do it in Java 8?

You want to implement parallel processing on collections. How can streams help?

You need to remove null-check boilerplate from your code. How can Optional help?

How can you handle default behavior in interfaces without breaking existing implementations?

You want to provide utility methods in interfaces. How would you do it?

How do you implement a simple event dispatcher using Java 8 functional interfaces?

You want to reduce lambda verbosity in common operations. How can method references help?

You need to implement supplier-based lazy evaluation. Which Java 8 interface would you use?

You want to create reusable comparators in a functional style. How would you do it?

You need to implement consumer-based logging for collections. Which interface would you use?

You want to compose multiple functions in a readable way. How can Function interface help?

You want to avoid boilerplate for Runnable or Callable implementations. How would lambdas help?

You need to perform map-reduce operations. How do streams help in this scenario?

You want to process large datasets concurrently. How do parallel streams help?

You want to transform collections into different data structures. How can streams assist?

You want to debug functional operations on streams effectively. How would you approach it?

2. Lambda Expressions & Functional Interfaces (41–80)

You need to implement a simple predicate to filter data. How would you use Java 8 lambdas?

How would you create a reusable comparator using lambdas?

You want to chain multiple consumers. How can you achieve it?

How would you implement a supplier-based random number generator?

You need to pass behavior as a method parameter. Which functional interface would you use?

You want to combine multiple predicates dynamically. How would you do it?

How would you implement a function composition for data transformation?

You need to implement a callback mechanism using functional interfaces.

You want to replace anonymous classes in event handling with lambdas.

You need a function that takes input and returns output. Which interface should you use?

You want to implement side-effect operations on collections. Which interface is suitable?

You need a supplier that generates unique IDs lazily.

How can you reuse functional interfaces in different parts of your application?

You want to compose functions in a pipeline. How would you implement it?

How would you handle optional behavior in functional interfaces?

You need a function that consumes a value without returning anything.

You want to implement a simple strategy pattern using lambdas.

How do you use method references to simplify lambda expressions?

How do you implement conditional execution using functional interfaces?

You need to convert one object type to another using a functional interface.

You want to cache results of a function. How would you implement it using Java 8?

You want to process collections using custom functional logic.

You need a function to wrap exceptions in functional pipelines.

How do you implement a factory pattern using suppliers?

You want to create reusable logging logic using consumers.

How do you handle null-safe function composition?

You want to execute multiple consumers sequentially. How would you do it?

How do you implement memoization for functions in Java 8?

You need to pass lambda expressions as arguments to methods.

How do you implement dynamic filtering using predicates?

How do you chain multiple functions in streams?

You need a functional interface for event handling.

How do you implement lazy initialization using suppliers?

You want to use lambdas to replace inner classes for comparators.

How do you handle exceptions inside lambdas?

You need to create higher-order functions using Java 8.

How do you implement a custom functional interface for your business logic?

How do you use BiFunction for mapping two inputs to one output?

You want to implement conditional execution using Predicate and Consumer.

How do you test lambda-based code effectively?

3. Streams API (81–120)

You need to filter a list of employees based on salary using streams. How would you do it?

How do you sort a collection dynamically with streams?

You need to perform a map operation to convert objects to another type.

How do you implement flatMap for nested collections?

You want to perform reduce operations on a list of integers.

How do you count elements satisfying a condition using streams?

How do you implement findFirst and findAny operations?

You want to collect results into a Set instead of a List. How would you do it?

How do you group elements by a property using streams?

How do you partition elements based on a predicate?

You need to calculate sum, average, min, and max using streams.

How do you implement distinct elements filtering?

You want to implement parallel streams for performance. How do you do it safely?

How do you handle null elements in streams?

How do you implement limit and skip operations in streams?

You want to flatten a list of lists into a single list using streams.

How do you implement joining of string elements from a collection?

You need to collect elements into a map using streams.

How do you implement custom collectors?

How do you debug streams operations step-by-step?

You want to perform conditional mapping of elements. How would you do it?

How do you implement multi-level grouping using streams?

You want to implement summarizing statistics on numeric streams.

How do you combine multiple stream operations efficiently?

You want to filter and transform elements in a single pipeline.

How do you handle checked exceptions in stream operations?

How do you implement peek for debugging or side-effects?

How do you merge two streams into one?

You want to process a stream of objects concurrently. How would you ensure thread-safety?

How do you implement minBy and maxBy custom comparator collectors?

How do you implement reducing operations with initial values?

You want to convert a stream into an array efficiently.

How do you implement stream concatenation with multiple sources?

How do you implement takeWhile and dropWhile in streams?

How do you implement infinite streams using Stream.generate or Stream.iterate?

How do you handle resource management for streams from I/O sources?

You want to collect elements into immutable collections. How would you do it?

How do you implement conditional filtering using predicates?

You want to implement distinct mapping by a key property. How would you do it?

How do you implement multi-criteria sorting using streams?

4. Optional & Date/Time API (121–160)

You want to avoid NullPointerExceptions when accessing nested objects. How would Optional help?

How do you implement default values with Optional?

How do you check presence or absence of a value in Optional?

You want to chain multiple Optional operations. How do you do it?

How do you convert Optional to Stream for further processing?

How do you implement mapping of Optional values?

How do you implement flatMap for Optional?

You want to throw custom exceptions if Optional is empty.

How do you safely extract a value from Optional without null checks?

How do you implement filtering of Optional values?

You need to calculate the duration between two LocalDateTimes.

How do you get the current date and time using Java 8 Date/Time API?

How do you convert legacy Date to LocalDate or LocalDateTime?

How do you format LocalDateTime into a custom string pattern?

How do you parse a string into LocalDateTime?

How do you implement time zone conversions with ZonedDateTime?

You need to calculate period or duration between two dates.

How do you add or subtract days, months, or years from a date?

How do you compare two LocalDates or LocalDateTimes?

How do you implement recurring events using Java 8 Date/Time API?

How do you convert Instant to LocalDateTime and vice versa?

How do you implement duration-based calculations?

How do you handle daylight saving changes when calculating time differences?

How do you create a LocalDate representing the first or last day of a month?

How do you create a stream of dates between two LocalDates?

How do you calculate age based on birth date using Period?

How do you extract year, month, or day from a LocalDate?

How do you implement formatting and parsing for multiple locales?

How do you use TemporalAdjusters for date manipulation?

How do you implement business logic based on working days using Java 8 Date API?

How do you combine Optional with streams for safe data processing?

How do you chain Optional, map, and filter for nested objects?

How do you handle missing values in Optional while mapping to other types?

How do you implement conditional execution with Optional?

How do you implement fallback values in Optional?

How do you convert Optional to nullable values for legacy code?

How do you implement safe navigation of nested collections using Optional?

How do you implement mapping Optional<Optional<T>> to Optional<T>?

How do you integrate Optional with stream pipelines effectively?

How do you implement functional error handling using Optional?

5. Concurrency Enhancements & Miscellaneous (161–200)

You need to process a large collection concurrently. How do parallel streams help?

How do you avoid race conditions when using parallel streams?

How do you combine parallel stream results safely?

How do you optimize parallel stream performance?

How do you handle thread-safety with shared resources in streams?

How do you implement ForkJoinPool with parallel streams?

How do you debug parallel stream operations?

You need to implement a non-blocking asynchronous computation. How can CompletableFuture help?

How do you chain multiple CompletableFuture operations?

How do you handle exceptions in CompletableFuture pipelines?

How do you combine results of multiple CompletableFutures?

How do you implement timeouts for CompletableFutures?

How do you integrate streams with CompletableFuture for async processing?

How do you implement supplyAsync or runAsync for background tasks?

How do you implement thenApply, thenAccept, and thenCompose for functional chaining?

How do you implement parallel task execution with CompletableFuture?

How do you handle fallback logic in asynchronous computations?

How do you implement lazy initialization safely using suppliers?

How do you implement memoization for expensive functions?

How do you implement recursion with streams or lambdas?

How do you design immutable data structures using Java 8 features?

How do you implement functional composition for business logic?

How do you debug complex lambda and stream pipelines?

How do you optimize memory usage with large streams?

How do you implement multi-threaded filtering or mapping?

How do you convert a blocking API to a non-blocking pipeline using Java 8?

How do you implement custom collectors for parallel streams?

How do you implement type-safe functional interfaces for generic operations?

How do you handle optional and default values in parallel streams?

How do you implement conditional execution in CompletableFuture pipelines?

How do you implement batch processing using streams and lambdas?

How do you integrate legacy collection APIs with Java 8 streams?

How do you implement logging in parallel stream operations without race conditions?

How do you profile performance of complex stream and lambda pipelines?

How do you implement retry logic for asynchronous tasks using CompletableFuture?

How do you implement fan-out/fan-in patterns using Java 8 concurrency features?

How do you safely share mutable state across parallel streams?

How do you implement functional pipelines for real-time data processing?

How do you integrate Optional, streams, and CompletableFuture for resilient code?

How do you write testable, maintainable, and concurrent code using Java 8 features?

##
##
##
##

1. Java 8 Basics & Features (1–40)

What are the main features introduced in Java 8?

What is a default method in an interface?

What is a static method in an interface?

What are functional interfaces?

Explain method references in Java 8.

What is the difference between a lambda expression and an anonymous class?

What are the advantages of lambda expressions?

What is type inference in Java 8?

What is the purpose of the Optional class?

What is a Stream in Java 8?

What are intermediate and terminal operations in streams?

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

What is the difference between map() and flatMap()?

How does Java 8 handle multiple inheritance in interfaces?

What is the significance of the java.util.function package?

What is the difference between Predicate, Function, Consumer, and Supplier?

What is the difference between sequential and parallel streams?

What is the difference between Iterator and Spliterator?

What are the benefits of Streams over Collections?

What is a Collector in Java 8?

Explain the difference between Optional.of() and Optional.ofNullable().

What is the difference between LocalDate, LocalTime, and LocalDateTime?

What is the purpose of the ZonedDateTime class?

Explain the difference between Duration and Period.

What is the significance of the java.time package?

What are the enhancements in Java 8 for Date/Time API?

What is the difference between removeIf() and filter() in streams?

What is the difference between anyMatch(), allMatch(), and noneMatch()?

What is the purpose of the findFirst() and findAny() methods?

What is the difference between count() and reduce() in streams?

Explain the significance of Comparator.comparing() in Java 8.

What are the new default methods added in Collection interfaces?

What is the difference between Stream.of() and Arrays.stream()?

What is the difference between limit() and skip() in streams?

What is the purpose of the forEachOrdered() method?

Explain the difference between Collection.stream() and
Collection.parallelStream().

What are the differences between Comparable and Comparator in Java 8?

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

What are the advantages of using method references over lambdas?

What is the difference between Optional.get() and Optional.orElse()?

2. Lambda Expressions & Functional Interfaces (41–80)

What is a lambda expression in Java 8?

What are the types of lambda expressions?

How do you define a functional interface?

Can a functional interface have multiple default methods?

Can a functional interface extend another interface?

How do you use a lambda expression as an argument to a method?

What are the rules for lambda parameters?

Can lambdas throw checked exceptions?

Can lambdas access local variables?

What is the difference between a lambda and an anonymous inner class?

Can lambdas capture this and super references?

What are the advantages of using functional interfaces?

What is the difference between Predicate and Function?

What is the difference between Consumer and Supplier?

How do you chain Predicates or Functions?

What is the use of UnaryOperator and BinaryOperator?

What is the difference between Function.andThen() and Function.compose()?

How do you implement a strategy pattern using lambda?

How do you implement a callback using lambda expressions?

What is the difference between lambda expressions and method references?

How do you implement lazy evaluation using Supplier?

How do you implement side-effects using Consumer?

Can functional interfaces have static methods?

Can functional interfaces have default methods?

What is the significance of @FunctionalInterface annotation?

How do you implement higher-order functions using lambdas?

What is the difference between BiFunction and BinaryOperator?

What is the difference between BiConsumer and Consumer?

How do you compose Consumers for sequential execution?

How do you compose Predicates for complex conditions?

How do you convert a lambda to a method reference?

What is the difference between explicit and inferred lambda types?

How do you handle exceptions inside lambda expressions?

How do you implement memoization with lambdas?

What is the role of lambda expressions in functional programming?

How do lambdas improve code readability and maintainability?

How do you use lambdas with event listeners?

Can a lambda implement multiple functional interfaces simultaneously?

How do lambdas differ from anonymous inner classes in memory usage?

What are the limitations of lambda expressions?
3. Streams API (81–120)

What is a Stream in Java 8?

What are the types of streams in Java 8?

What is the difference between sequential and parallel streams?

What are intermediate and terminal operations in streams?

What is the difference between map() and flatMap()?

How do you filter elements in a stream?

How do you sort elements in a stream?

How do you use limit() and skip() methods?

What is the difference between forEach() and forEachOrdered()?

How do you use reduce() in streams?

How do you use collect() in streams?

What is the difference between collect() and reduce()?

How do you use groupingBy() collector?

How do you use partitioningBy() collector?

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

How do you convert a stream to a list?

How do you convert a stream to a set?

How do you convert a stream to a map?

How do you flatten nested collections using flatMap()?

How do you find the first element in a stream?

How do you find any element in a stream?

How do you check if anyMatch(), allMatch(), or noneMatch() conditions are satisfied?

How do you count elements in a stream?

How do you find max() and min() in streams?

What are primitive streams in Java 8?

What is the difference between IntStream, LongStream, and DoubleStream?

How do you generate infinite streams?

How do you create a stream from an array?

How do you create a stream from a collection?

How do you merge two streams?

How do you implement takeWhile() and dropWhile()?

How do you implement distinct() in streams?

How do you implement stream concatenation?

How do you implement parallel streams?

How do you ensure thread safety with parallel streams?

How do you use peek() for debugging in streams?

How do you use Stream.of() vs Arrays.stream()?

How do you implement map-reduce operations?

How do you implement conditional filtering in streams?

How do you handle checked exceptions in stream operations?

4. Optional & Date/Time API (121–160)

What is the purpose of the Optional class?

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

How do you get the value from an Optional?

How do you provide a default value using Optional?

How do you check if an Optional is empty or present?

How do you chain map() and filter() in Optional?

How do you use flatMap() with Optional?

How do you throw an exception if Optional is empty?

How do you convert Optional to Stream?

How do you avoid NullPointerException with Optional?

What is the difference between LocalDate, LocalTime, and LocalDateTime?

How do you get the current date and time?

How do you parse a string into LocalDate or LocalDateTime?

How do you format LocalDate or LocalDateTime?

How do you add or subtract days, months, or years from a date?

How do you compare two LocalDates or LocalDateTimes?

What is ZonedDateTime and why is it used?

How do you convert Instant to LocalDateTime?

How do you calculate duration between two LocalDateTimes?

How do you calculate period between two LocalDates?

What is the difference between Duration and Period?

How do you get year, month, and day from a LocalDate?

How do you get hour, minute, and second from a LocalTime?

How do you implement TemporalAdjusters?

How do you handle time zone conversions?

How do you create a stream of dates?

How do you calculate age from birthdate?

How do you implement recurring events with LocalDate?

How do you handle daylight saving changes in calculations?

How do you convert Date to LocalDate or LocalDateTime?

How do you convert LocalDateTime to Date?

How do you implement safe navigation with Optional?

How do you implement conditional mapping with Optional?

How do you handle nested Optional values?

How do you provide fallback values with Optional?

How do you combine Optional with streams?

How do you implement mapping Optional<Optional<T>> to Optional<T>?

How do you handle null-safe operations in Optional?

How do you implement functional error handling using Optional?

How do you use Optional in method return types?

5. Concurrency, CompletableFuture & Misc (161–200)

What is a parallel stream?

How do parallel streams improve performance?

How do you ensure thread safety with parallel streams?

What is ForkJoinPool?

How do you use ForkJoinPool with streams?

How do you debug parallel streams?

What is CompletableFuture in Java 8?

How do you create a CompletableFuture?

What is supplyAsync() in CompletableFuture?

What is runAsync() in CompletableFuture?

How do you chain CompletableFutures?

How do you handle exceptions in CompletableFuture?

How do you combine multiple CompletableFutures?

What is thenApply() in CompletableFuture?

What is thenAccept() in CompletableFuture?

What is thenCompose() in CompletableFuture?

How do you implement timeouts in CompletableFuture?

How do you implement fallback logic in CompletableFuture?

How do you integrate CompletableFuture with streams?

How do you perform asynchronous computations safely?

How do you implement lazy initialization using Supplier?

How do you implement memoization with Supplier or Function?

How do you implement recursion using streams or lambdas?

How do you design immutable data structures with Java 8 features?

How do you compose functions for business logic?

How do you debug complex lambda and stream pipelines?

How do you optimize memory usage for large streams?

How do you implement multi-threaded filtering or mapping?

How do you convert a blocking API to non-blocking using CompletableFuture?

How do you implement custom collectors for streams?

How do you handle optional and default values in streams?

How do you implement batch processing using streams?

How do you integrate legacy APIs with Java 8 streams?

How do you implement logging in parallel stream operations?

How do you profile performance of complex streams and lambdas?

How do you implement retry logic in asynchronous tasks?

How do you implement fan-out/fan-in patterns with CompletableFuture?

How do you safely share mutable state across parallel streams?

How do you integrate Optional, streams, and CompletableFuture together?

How do you write testable and maintainable concurrent code in Java 8?

##
##
##
##

1. Spring Boot Basics & Configuration (1–40)

You need to quickly start a RESTful application. How would Spring Boot help?

You want to reduce XML configuration in your project. How does Spring Boot assist?

You need to externalize application properties. How would you use application.properties or application.yml?

You want to load different profiles (dev, test, prod). How would you achieve this in Spring Boot?

You need to configure beans automatically without manual registration. How does Spring Boot help?

You want to override auto-configuration for a specific bean. How would you do it?

You want to create a standalone Spring Boot application. What annotation would you use?

You need to expose a health check endpoint. How would you implement it?

You want to enable detailed startup logs for debugging. How would you configure it?

You need to configure default server port and context path. How would you do it?

You want to disable a specific auto-configuration class. How would you achieve this?

You want to implement custom startup logic. Which Spring Boot interface would you use?

You need to run code after the application context is initialized. How would you implement it?

You want to monitor application metrics. How would you configure Actuator endpoints?

You need to secure a REST endpoint with basic authentication. How would Spring Boot help?

You want to enable scheduling tasks. Which annotation would you use?

You need to read environment variables in your application. How would you do it?

You want to customize the banner displayed at startup. How would you achieve this?

You need to configure file upload limits. How would you do it?

You want to configure CORS globally for all endpoints. How would you implement it?

You need to configure exception handling globally. How would you do it?

You want to bind properties to POJO classes. How would you implement this?

You need to change logging levels dynamically. How would you do it?

You want to enable retry for failed service calls. How would Spring Boot help?

You need to implement graceful shutdown for your application. How would you do it?

You want to enable caching in your application. How would you configure it?

You want to configure multiple datasources in Spring Boot. How would you achieve this?

You need to read configuration from an external file outside your jar. How would you do it?

You want to configure default error handling for REST controllers. How would you implement it?

You want to configure asynchronous methods. How would you do it?

You need to customize Spring Boot’s embedded server. How would you achieve it?

You want to configure SSL for your Spring Boot application. How would you do it?

You need to set a global exception handler for your REST API. How would you do it?

You want to enable JMX support in Spring Boot. How would you do it?

You need to implement a command-line runner to execute logic at startup. How would you do it?

You want to configure file-based logging. How would you achieve this?

You need to load configuration dynamically without restarting the app. How would you implement it?

You want to configure Actuator endpoints for production securely. How would you do it?

You need to implement custom health indicators. How would you achieve this?

You want to monitor JVM metrics in Spring Boot. How would you do it?

2. Spring Boot REST & Controllers (41–80)

You want to create a REST API to fetch user data. How would you design the controller?

You need to handle POST requests with JSON payload. How would you implement it?

You want to validate incoming request data. How would you do it?

You need to implement global exception handling for REST APIs. How would you achieve it?

You want to send custom HTTP status codes. How would you do it?

You need to implement HATEOAS in your API. How would you achieve this?

You want to version your API endpoints. How would you implement it?

You need to restrict access to certain endpoints. How would you do it?

You want to return paginated responses. How would you implement pagination in Spring Boot?

You need to implement filtering of REST responses. How would you do it?

You want to compress REST responses. How would you achieve it?

You need to handle cross-origin requests (CORS). How would you implement it?

You want to implement API rate limiting. How would Spring Boot help?

You need to implement async REST calls. How would you achieve this?

You want to handle file uploads in a REST API. How would you implement it?

You need to handle file downloads in a REST API. How would you implement it?

You want to implement a versioned response body. How would you do it?

You need to implement caching for a REST API. How would you achieve this?

You want to handle optional query parameters. How would you implement it?

You need to implement custom response headers. How would you achieve this?

You want to use DTOs for request and response. How would you implement it?

You need to map nested objects in request payloads. How would you do it?

You want to implement a REST API with HATEOAS links. How would you do it?

You need to implement exception translation for database errors. How would you do it?

You want to implement content negotiation (JSON, XML). How would you achieve it?

You need to log all incoming REST requests. How would you implement it?

You want to return standardized error responses. How would you do it?

You need to handle missing request parameters gracefully. How would you implement it?

You want to secure REST endpoints with JWT. How would you implement it?

You need to implement API documentation using Swagger. How would you do it?

You want to use Spring Boot DevTools for hot reloading REST controllers. How would you do it?

You need to implement custom serialization for response objects. How would you do it?

You want to implement rate-limiting per user. How would you do it?

You need to implement exception handling for async REST methods. How would you achieve it?

You want to implement controller advice for centralized exception handling. How would you do it?

You need to bind query parameters to POJO objects. How would you implement it?

You want to implement path variable validation. How would you do it?

You need to implement custom annotations for REST endpoints. How would you do it?

You want to implement response caching. How would you achieve this?

You need to implement pagination and sorting in REST API using Spring Data. How would you do it?

If you want, I can continue and create the remaining 120 scenario-based Spring Boot questions (for Data, JPA, Security, Actuator, Testing, Microservices integration, etc.) so we have a full 200-question set.

Do you want me to continue with the rest?

You said:
200
ChatGPT said:

Perfect! Here’s a complete set of 200 scenario-based Spring Boot interview questions, organized topic-wise for easy reference. I’ll continue from where we left off.

3. Spring Boot Data & JPA (81–120)

You want to configure Spring Data JPA in your project. How would you do it?

You need to define a custom repository method. How would you implement it?

You want to use query derivation to fetch data. How would you achieve it?

You need to write a JPQL query for a custom operation. How would you do it?

You want to use native SQL queries in Spring Data JPA. How would you implement them?

You need to handle pagination in database queries. How would you implement it?

You want to sort query results dynamically. How would you achieve this?

You need to implement a one-to-many relationship. How would you do it?

You need to implement a many-to-many relationship. How would you do it?

You want to fetch related entities lazily. How would you configure it?

You want to fetch related entities eagerly. How would you configure it?

You need to handle transactions programmatically. How would you implement it?

You want to use @Transactional annotation effectively. How would you do it?

You need to handle rollback in case of exceptions. How would you implement it?

You want to implement optimistic locking. How would you configure it?

You want to implement pessimistic locking. How would you configure it?

You need to batch insert records efficiently. How would you implement it?

You want to map entity fields to different column names. How would you do it?

You need to implement soft delete in your entities. How would you do it?

You want to automatically generate primary keys. How would you configure it?

You need to implement auditing (createdBy, createdDate, updatedBy, updatedDate). How would you do it?

You want to use projection to fetch partial data. How would you implement it?

You need to implement DTO mapping from entities. How would you do it?

You want to implement custom repository behavior. How would you achieve it?

You need to handle native queries with pagination. How would you do it?

You want to integrate Spring Data JPA with QueryDSL. How would you implement it?

You need to configure multiple datasources with JPA. How would you do it?

You want to implement soft deletes using @SQLDelete and @Where annotations. How would you do it?

You need to implement entity validation with JSR-303 annotations. How would you do it?

You want to handle complex joins using Spring Data JPA. How would you achieve it?

You need to implement bulk update/delete queries. How would you do it?

You want to log executed SQL queries. How would you configure it?

You need to implement a read-only repository method. How would you do it?

You want to implement caching of JPA queries. How would you achieve it?

You need to handle N+1 select problem. How would you resolve it?

You want to implement a custom dialect for your database. How would you do it?

You need to handle database migrations. How would you implement it using Flyway or Liquibase?

You want to implement query hints for performance optimization. How would you do it?

You need to fetch entities with dynamic filters. How would you implement it?

You want to integrate Spring Data JPA with specifications for dynamic queries. How would you achieve it?

4. Spring Boot Security & Authentication (121–160)

You want to secure your REST endpoints with Spring Security. How would you do it?

You need to implement basic authentication. How would you configure it?

You want to implement JWT-based authentication. How would you do it?

You need to implement OAuth2 login. How would you configure it?

You want to restrict access to endpoints based on roles. How would you implement it?

You need to implement method-level security. How would you achieve it?

You want to encrypt passwords. How would you configure password encoding?

You need to implement login failure handling. How would you do it?

You want to implement token expiration handling. How would you achieve it?

You need to implement refresh token mechanism. How would you do it?

You want to configure CORS with Spring Security. How would you implement it?

You need to handle access denied exceptions. How would you configure it?

You want to log security events. How would you implement it?

You need to integrate Spring Security with OAuth2 providers like Google or Facebook. How would you do it?

You want to implement custom authentication provider. How would you achieve it?

You need to implement stateless authentication for REST APIs. How would you configure it?

You want to secure endpoints selectively based on URL patterns. How would you do it?

You need to implement CSRF protection. How would you configure it?

You want to implement remember-me functionality. How would you do it?

You need to implement two-factor authentication. How would you configure it?

You want to implement logout handling with Spring Security. How would you achieve it?

You need to configure session management. How would you do it?

You want to implement custom access decision logic. How would you do it?

You need to implement multi-role based access control. How would you configure it?

You want to integrate Spring Security with LDAP. How would you implement it?

You need to implement password reset functionality. How would you achieve it?

You want to implement IP-based access restriction. How would you configure it?

You need to handle security for file upload endpoints. How would you do it?

You want to implement authorization based on user permissions. How would you configure it?

You need to implement security for WebSocket endpoints. How would you do it?

You want to configure Spring Security without XML. How would you achieve it?

You need to handle authentication exceptions globally. How would you do it?

You want to implement custom security filters. How would you configure them?

You need to implement JWT token validation for REST APIs. How would you do it?

You want to implement stateless authentication with API keys. How would you achieve it?

You need to implement role hierarchy in Spring Security. How would you configure it?

You want to integrate Spring Security with OAuth2 Resource Server. How would you do it?

You need to implement authorization checks for method parameters. How would you do it?

You want to integrate Spring Security with social login providers. How would you do it?

You need to implement custom UserDetailsService. How would you achieve it?

5. Spring Boot Actuator, Testing & Misc (161–200)

You want to monitor application health using Spring Boot Actuator. How would you do it?

You need to expose custom Actuator endpoints. How would you implement them?

You want to secure Actuator endpoints in production. How would you configure it?

You need to monitor application metrics. How would you achieve it?

You want to track HTTP request metrics. How would you configure it?

You need to monitor database connection pools. How would you achieve it?

You want to implement logging levels dynamically using Actuator. How would you do it?

You need to track JVM metrics using Actuator. How would you implement it?

You want to implement integration tests for REST APIs. How would you do it?

You need to mock services in unit tests. How would you achieve it?

You want to test Spring Boot JPA repositories. How would you implement it?

You need to test REST controllers using MockMvc. How would you do it?

You want to implement end-to-end testing. How would you achieve it?

You need to handle database rollback after tests. How would you implement it?

You want to test exception handling in REST APIs. How would you do it?

You need to test asynchronous methods. How would you achieve it?

You want to test security configurations in Spring Boot. How would you implement it?

You need to test application properties loading. How would you do it?

You want to test configuration of multiple profiles. How would you achieve it?

You need to test caching functionality. How would you implement it?

You want to implement logging tests for REST requests. How would you achieve it?

You need to test file upload endpoints. How would you implement it?

You want to test scheduled tasks. How would you achieve it?

You need to implement health check tests for Actuator endpoints. How would you do it?

You want to test database migrations using Flyway or Liquibase. How would you achieve it?

You need to test dynamic configuration reload. How would you implement it?

You want to test custom exception handling globally. How would you achieve it?

You need to test API versioning. How would you implement it?

You want to test HATEOAS responses. How would you do it?

You need to implement contract testing for REST APIs. How would you achieve it?

You want to test JWT authentication flows. How would you implement it?

You need to test role-based access control. How would you achieve it?

You want to test OAuth2 authentication. How would you implement it?

You need to test multi-datasource configurations. How would you achieve it?

You want to test custom Spring Boot starters. How would you implement it?

You need to test integration with external APIs. How would you achieve it?

You want to test WebSocket endpoints. How would you implement it?

You need to test batch jobs configured in Spring Boot. How would you achieve it?

You want to monitor real-time metrics during integration tests. How would you do it?

You need to ensure your Spring Boot application is production-ready. What checks would you perform?

✅ That completes 200 scenario-based Spring Boot interview questions.

If you want, I can also prepare Microservices scenario-based questions next, with a full 200-question set.

Do you want me to do that?

##
##

##
##