**Java & Spring Boot Interview Questions - Detailed Answers**

**1. Difference between ArrayList and LinkedList**

* **ArrayList**:
  + Backed by a **dynamic array**.
  + **Fast access** using index (O(1) time for get/set).
  + **Slow insert/delete** in the middle because it shifts elements (O(n)).
  + Better for frequent reads.
* **LinkedList**:
  + Backed by a **doubly linked list**.
  + **Fast insert/delete** at any position (O(1) if pointer known).
  + **Slow access** (O(n) time for get/set).
  + Better for frequent inserts/deletes.

**2. ExecutorService**

* Part of the java.util.concurrent package.
* Manages a pool of threads.
* Submit tasks using submit() or execute().
* Can return Future objects to retrieve results asynchronously.
* Example:

ExecutorService executor = Executors.newFixedThreadPool(2);

executor.submit(() -> System.out.println("Task executed"));

executor.shutdown();

**3. Runnable vs Callable**

* **Runnable**:
  + No return value.
  + Cannot throw checked exceptions.
  + Example:
  + Runnable task = () -> System.out.println("Running");
* **Callable**:
  + Returns a value.
  + Can throw checked exceptions.
  + Example:
  + Callable<String> task = () -> "Result";

**4. ConcurrentHashMap**

* A thread-safe map that allows concurrent read and write.
* Uses **segment locking** (Java 7) or **bucket-level locking** (Java 8).
* Prevents ConcurrentModificationException.
* Efficient alternative to Collections.synchronizedMap().

**5. Comparable vs Comparator**

* **Comparable**:
  + Used for natural sorting.
  + Implemented inside the class.
  + Example: public int compareTo(Object o)
* **Comparator**:
  + Used for custom sorting.
  + External logic.
  + Example: compare(Employee a, Employee b)

**6. How to Design a REST Web Service**

1. Define resource names (e.g., /users, /orders).
2. Use correct HTTP verbs: GET, POST, PUT, DELETE.
3. Create controller using @RestController.
4. Use annotations like @GetMapping, @PostMapping, etc.
5. Accept and return JSON using @RequestBody and @ResponseBody.
6. Validate input using @Valid.
7. Add exception handling using @ControllerAdvice.

**7. application.properties vs application.yml**

* **application.properties**:
  + Key-value pair format.
  + Flat and easy to use.
* **application.yml**:
  + YAML format.
  + Supports hierarchy and nesting.
* **Precedence**: application.properties overrides .yml.

**8. Microservices Architecture**

* A way to build an application as a collection of small, independent services.
* Each service runs in its own process and communicates via APIs.
* Benefits:
  + Scalability
  + Independent deployments
  + Resilience

**9. Microservices Design Patterns**

* **API Gateway**: Single entry point.
* **Service Registry (Eureka)**: Register and discover services.
* **Circuit Breaker (Resilience4J)**: Prevent system overload.
* **Bulkhead**: Isolate failures.
* **Saga**: Manage distributed transactions.

**10. IntStream - range vs rangeClosed**

* range(start, end) – excludes the end.
* rangeClosed(start, end) – includes the end.
* Example:
* IntStream.range(1, 5); // 1, 2, 3, 4
* IntStream.rangeClosed(1, 5); // 1, 2, 3, 4, 5

**11. IntStream**

* A specialized stream for int primitives.
* Efficient operations like sum(), average(), map(), etc.

**12. Spring vs Spring Boot**

* **Spring**:
  + Requires manual configuration.
  + More boilerplate code.
* **Spring Boot**:
  + Auto-configuration.
  + Embedded server (Tomcat).
  + Easier to set up.

**13. Dependency Injection**

* Process of injecting dependent objects instead of creating them.
* Promotes loose coupling.
* Spring uses @Autowired, constructor injection, or setter injection.

**14. Same Bean Name in Spring Boot**

* Causes a conflict.
* Spring throws NoUniqueBeanDefinitionException.
* Resolve using:
  + @Primary
  + @Qualifier("beanName")

**15. Anonymous Class in Java**

* A class without a name used to create one-time use classes.
* Typically used in event handling or thread creation.
* Example:

Runnable r = new Runnable() {

public void run() {

System.out.println("Running");

}

};

**16. Reverse String using IntStream**

String input = "hello";

String reversed = IntStream.range(0, input.length())

.mapToObj(i -> input.charAt(input.length() - 1 - i))

.map(String::valueOf)

.collect(Collectors.joining());

System.out.println(reversed); // "olleh"

**17. SQL Queries**

* **Create Table**:

CREATE TABLE employees (id INT, name VARCHAR(50), dept VARCHAR(30));

* **Insert Data**:

INSERT INTO employees VALUES (1, 'John', 'HR');

* **Group By**:

SELECT dept, COUNT(\*) FROM employees GROUP BY dept;

**18. JUnit Annotations**

* @Test: Marks a test method.
* @BeforeEach: Runs before each test method.
* @AfterEach: Runs after each test.
* @BeforeAll: Runs once before all tests (static).
* @AfterAll: Runs once after all tests.

**19. Bubble Sort Algorithm**

int[] arr = {5, 3, 8, 4};

for (int i = 0; i < arr.length - 1; i++) {

for (int j = 0; j < arr.length - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

**20. Proving String is Immutable**

String s = "hello";

String s2 = s;

s = s.concat(" world");

System.out.println(s2); // prints "hello"

* Original s value doesn't change, showing immutability.

**21. @Transactional and @Conditional**

* **@Transactional**:
  + Wraps method in a database transaction.
  + Rolls back on exceptions.
* **@Conditional**:
  + Registers beans conditionally based on property or class.

**22. Collections: Duplicates & Insertion Order**

* **List**:
  + Allows duplicates.
  + Maintains insertion order.
* **Set**:
  + No duplicates.
  + HashSet: Unordered.
  + LinkedHashSet: Maintains insertion order.
  + TreeSet: Sorted, no duplicates.

**✅ Internal Working of HashMap**

**Answer:**  
HashMap stores data in key-value pairs. Internally, it uses an array of buckets, each being a linked list or a tree (if many entries hash to the same bucket). The key’s hashCode() is used to calculate the index in the array, and equals() checks for key equality in case of collisions.

**Explanation:**  
If two keys have the same hash, they go into the same bucket (collision). In Java 8+, if collisions exceed a threshold (8 entries), the list turns into a balanced tree for faster lookup.

**✅ Internal Working of ConcurrentHashMap**

**Answer:**  
ConcurrentHashMap divides the map into segments (before Java 8) or uses bucket-level locks (Java 8+). It avoids locking the entire map and only locks specific segments or bins during updates, allowing high concurrency.

**Explanation:**  
In Java 8+, it uses a combination of CAS (Compare-And-Swap) and fine-grained locking to ensure thread safety with better performance than synchronized maps.

**✅ Can we make HashMap synchronized? How?**

**Answer:**  
Yes, use Collections.synchronizedMap(new HashMap<>()) to wrap a HashMap with synchronized access.

**Explanation:**  
This provides thread safety by synchronizing all method calls. However, it blocks all threads during write or read operations, which may degrade performance.

**✅ Difference Between ConcurrentHashMap and SynchronizedMap**

| **Feature** | **ConcurrentHashMap** | **SynchronizedMap** |
| --- | --- | --- |
| Thread-safety | Yes, with fine-grained locking | Yes, via synchronized methods |
| Performance | Better, due to segmented locks | Slower, global lock on entire map |
| Null key/value | Does not allow null key/value | Allows one null key and multiple null values |
| Iteration | Weakly consistent | Fail-fast |

**✅ Internal Working of Set**

**Answer:**  
HashSet is the most common Set implementation and uses a HashMap internally where each element is stored as a key with a dummy value.

**Explanation:**  
Since keys in a HashMap are unique, the Set automatically ensures no duplicate elements.

**✅ Difference Between HashMap and Hashtable**

| **Feature** | **HashMap** | **Hashtable** |
| --- | --- | --- |
| Thread-safe | No | Yes (synchronized methods) |
| Performance | Faster | Slower |
| Null keys/values | Allows one null key, many null values | Doesn’t allow null keys/values |
| Legacy? | No | Yes (legacy) |

**✅ TreeMap and Internal Implementation**

**Answer:**  
TreeMap stores data in a Red-Black Tree, maintaining keys in sorted (natural or comparator-defined) order.

**Explanation:**  
All operations like get, put, and remove take O(log n) time due to balanced binary tree structure.

**✅ What is Synchronization?**

**Answer:**  
Synchronization is a mechanism to control access to shared resources in a multithreaded environment.

**Explanation:**  
It ensures that only one thread can access a block of code or method at a time to prevent race conditions.

**✅ What are Locks in Multithreading?**

**Answer:**  
Locks (from java.util.concurrent.locks) provide more flexible and granular thread control than synchronized blocks.

**Explanation:**  
Unlike synchronized methods, locks allow features like tryLock(), timed lock, and interruptible lock acquisition.

**✅ Difference Between Synchronized and ReentrantLock**

| **Feature** | **Synchronized** | **ReentrantLock** |
| --- | --- | --- |
| Lock type | Intrinsic (monitor lock) | Explicit object from java.util.concurrent |
| Try lock support | No | Yes |
| Fairness | No | Yes (can set fair policy) |
| Interruptible | No | Yes |
| Performance | Less flexible | More flexible and powerful |

**✅ How Reflection is Done in Java?**

**Answer:**  
Java Reflection API allows inspection and modification of classes, methods, fields, and constructors at runtime using classes in java.lang.reflect.

**Example:**

Class<?> cls = Class.forName("com.example.MyClass");

Method method = cls.getDeclaredMethod("myMethod");

method.setAccessible(true);

method.invoke(instance);

**✅ What is Circular Dependency?**

**Answer:**  
A circular dependency occurs when two or more beans are dependent on each other in a way that forms a cycle, making Spring unable to resolve dependencies.

**Example:**  
Class A depends on B, and B depends on A.

**Solution:**  
Use @Lazy, setter injection, or @PostConstruct method to break the cycle.

**✅ Spring Boot Lifecycle**

1. **Application starts**: main() method calls SpringApplication.run()
2. **Bootstrapping**: Application context is created.
3. **Component scan**: Scans and registers beans.
4. **Auto-configuration**: Applies default configurations.
5. **ApplicationRunner / CommandLineRunner**: Executes custom code.
6. **Application ready**: App is up and running.
7. **Shutdown hook**: Cleans up resources on shutdown.

**✅ Design Patterns Used in Spring Boot**

* **Singleton**: Default bean scope
* **Factory**: Bean instantiation
* **Proxy**: AOP features
* **Template Method**: JdbcTemplate, RestTemplate
* **Observer**: ApplicationEventPublisher
* **Builder**: Fluent APIs
* **Dependency Injection**: Core of Spring

**Multithreading Frameworks in Java**

* **Executor Framework**: Manages thread creation and execution with ExecutorService, ThreadPoolExecutor, etc.
* **Fork/Join Framework**: Efficiently splits tasks into subtasks using ForkJoinPool.
* **CompletableFuture**: Allows asynchronous, non-blocking programming with chaining and combining tasks.
* **ScheduledExecutorService**: Schedules tasks to run at fixed rate or delay.

**✅ What is Stream in Java?**

**Answer:** Streams in Java provide a high-level abstraction to process sequences of elements (usually from collections) in a functional style.

**Example:**

List<String> names = list.stream().filter(n -> n.startsWith("A")).collect(Collectors.toList());

**Types:**

* Sequential Stream
* Parallel Stream

**✅ Checked vs Unchecked Exception**

|  |  |  |
| --- | --- | --- |
| Type | Example | Handled at Compile Time? |
| Checked | IOException, SQLException | Yes |
| Unchecked | NullPointerException, ArithmeticException | No |

**Explanation:** Checked exceptions must be declared or handled. Unchecked are usually due to programming errors.

**✅ try, catch, throw, throws, finally**

* **try**: Encloses code that might throw an exception.
* **catch**: Handles the exception.
* **throw**: Used to explicitly throw an exception.
* **throws**: Declares exceptions that a method can throw.
* **finally**: Always executes after try/catch, used for cleanup.

**Efficiency:** Avoid using try-catch inside loops for performance. Use it only around risky blocks.

**✅ Can Static Method Be Overridden?**

**Answer:** No. Static methods are bound at compile time and belong to the class, not the instance. They can be hidden, not overridden.

**✅ What is Init Block?**

**Answer:** An instance initializer block is code inside {} outside any method, executed when an object is created.

**Example:**

{

System.out.println("Instance Block");

}

**✅ Can Constructor Call Another Constructor?**

**Answer:** Yes, using this() you can call one constructor from another in the same class. It must be the first statement.

**Example:**

public MyClass() {

this(10);

}

public MyClass(int x) {

// do something

}

**How to Monitor Microservices?**

* **Spring Boot Actuator**: Provides metrics, health checks, etc.
* **Prometheus + Grafana**: Collects and visualizes metrics.
* **ELK Stack (Elasticsearch, Logstash, Kibana)**: For log aggregation.
* **Zipkin/Sleuth**: For distributed tracing.
* **New Relic, Datadog**: Full observability tools.

**✅ Microservice Design Patterns**

* **API Gateway**: Single entry point for all services.
* **Circuit Breaker**: Prevents cascading failures.
* **Service Discovery**: Eureka/Consul for dynamic discovery.
* **Strangler Fig**: Gradual migration from monolith.
* **Bulkhead**: Isolates failures in parts of system.
* **Sidecar**: Adds capabilities without changing service.

**✅ Spring Boot Actuator**

Provides production-ready features:

* /actuator/health
* /actuator/metrics
* /actuator/info
* Enables endpoints for monitoring, metrics, and managing apps.

Add to pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

**✅ JaCoCo for Unit Testing**

**JaCoCo (Java Code Coverage)** measures how much code is tested by unit tests.

* Add plugin to pom.xml

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.8</version>

</plugin>

* Run mvn test + mvn jacoco:report

**✅ How to Test Private Methods in JUnit**

1. **Avoid if possible** — test via public methods.
2. If needed:
   * Use reflection:

Method method = MyClass.class.getDeclaredMethod("privateMethod");

method.setAccessible(true);

method.invoke(instance);

* Or restructure code to expose behavior via public methods.

**✅ What is @Mock and @InjectMocks?**

* @Mock: Creates a mock instance.
* @InjectMocks: Injects mocks into the tested class.

@Mock private UserRepository repo;

@InjectMocks private UserService service;

JUnit + Mockito uses them for unit tests.

**✅ What is Stubbing?**

Stubbing means configuring mock behavior:

when(mock.method()).thenReturn(value);

Used in unit testing to simulate responses.

**✅ What is Immutable Class? How to Create?**

A class whose state cannot change after creation.

**Steps to make immutable:**

* Make class final
* Make fields private final
* No setters
* Deep copy mutable objects
* Only getter methods

final class Employee {

private final String name;

public Employee(String name) { this.name = name; }

public String getName() { return name; }

}

**✅ How to Create Custom Annotation in Spring Boot**

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface MyCustomAnnotation {

String value();

}

Use reflection or AOP to process it.

**✅ Circuit Breaker, API Gateway Implementation**

* **Circuit Breaker**: Use Resilience4j

@CircuitBreaker(name = "myService", fallbackMethod = "fallback")

* **API Gateway**: Use Spring Cloud Gateway

spring:

cloud:

gateway:

routes:

- id: user-service

uri: lb://USER-SERVICE

predicates:

- Path=/users/\*\*

**✅ Pagination**

Use Pageable in Spring Data JPA:

Page<User> page = userRepository.findAll(PageRequest.of(0, 10));

Returns 10 records from page 0.

**✅ Sleuth**

Used for distributed tracing. Adds trace and span IDs to logs.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-sleuth</artifactId>

</dependency>

**✅ @Conditional Annotation**

Used to conditionally register beans:

@Conditional(OnWindowsCondition.class)

@Bean

public MyBean windowsBean() {...}

Can use @ConditionalOnProperty, @ConditionalOnMissingBean, etc.

**✅ application.properties vs application.yml: Which Has Higher Precedence?**

**Both have same precedence.** If both exist, application.properties overrides values in application.yml.

Use only one format to avoid confusion.