INTERVIW

- **VAISHANVI CHOPATE MOCK**

Of course. Here are all the answers provided by the interviewee, Vaishnavi, during the mock interview.

On Introduction & Background

- **"Can you please introduce yourself?"**
  - "My name is Vaishnavi Zopre. I am from Kanga. I have done my Bachelor's of Engineering in Computer Science."
- **"What are the things that you have covered in this one month?"**
  - "In this one month, I have covered the basics of Spring Boot, all the OOPs concepts, then how to connect with the database with the help of Spring Data JPA, and also some design patterns."

On Spring Boot & Database

- **"How can we perform database operations with Spring Boot?"**
  - "For performing database operations with Spring Boot, we use Spring Data JPA. Spring Data JPA provides a JPA repository for performing database operations, so with the

help of that, we can perform all the query-related operations with the database."

- **"Where do we store the database-specific username, password, and URL?"**
  - "We store it in application.properties."
- **"What do we have to do if we want to store it in a MySQL database?"**
  - "We just have to change the dependency and also the driver class name and the URL."
- **On Design Patterns**
- **"Are you aware of any design patterns?"**
  - "Yes, I'm aware of the Factory Design Pattern and the Template Design Pattern."
- **"Can you please explain the Template Design Pattern?"**
  - "The Template Design Pattern is basically used for code reusability. A base class is there which contains all the functionality, and all the functionalities are inherited by the child classes as per the requirement."
- **"And what about the Factory Design Pattern?"**
  - "The Factory Design Pattern is basically used to avoid tight coupling. Instead of creating an

object with the help of the new keyword, the factory is responsible for creating the object."

- **"So which is the factory in Spring Boot?"**
  - "The Application Context is the factory in Spring Boot."
- **On Core Java & OOPs Concepts**
- **"What are the OOPs concepts?"**
  - "The OOPs concepts are Polymorphism, Inheritance, Encapsulation, and Abstraction."
- **"Which is the parent of all the classes in Java?"**
  - "The Object class is the parent of all the classes in Java."
- **"Can you please explain polymorphism?"**
  - "Polymorphism means 'same name, different form'. Polymorphism is achieved with the help of method overriding and method overloading."
- **"What is the difference between method overriding and method overloading?"**
  - "Method overriding occurs in two different classes, which have a parent-child relationship. The method name and the input arguments should be the same, and the return type can be the same or covariant. The access

modifier can be the same or we can increase the scope, but we cannot decrease it. Method overloading occurs in the same class. The method name is the same, but the input arguments are different. The return type and the access modifier can be anything."

- **"Can we override a private method?"**
  - "No, we cannot override a private method."
- **"What is the use of the static keyword?"**
  - "The static keyword can be used at the variable level as well as at the method level. Static members are at the class level, so only one copy gets generated, and we can directly access them with the help of the class name."
- **"What is the use of the final keyword?"**
  - "The final keyword can be used at the variable level, method level, as well as at the class level."
  - **If we declare a class as final?** "Then we cannot create a child class of that class."
  - **If we declare a method as final? "**Then we cannot override that method."
  - **If we declare a variable as final?** "Then we cannot reassign a value to that variable."

- **"What is the difference between an interface and an abstract class?"**
  - "To declare an interface, we use the interface keyword, and for an abstract class, we use the abstract keyword. All the methods in an interface are by default public abstract, and all the variables are public static final. An abstract class can have abstract methods as well as concrete methods. To use an interface, we use the implements keyword, and for an abstract class, we use the extends keyword. An interface does not allow a constructor, while an abstract class allows a constructor."
- **"Can we create an object of an interface?"**
  - "No."
- **"Can we create an object of an abstract class?"**
  - "No."
- **"So, how will the constructor of the abstract class get called?"**
  - "With the help of the super() keyword from the child class constructor."
- **"What is inheritance?"**
  - "Inheritance is nothing but acquiring all the properties and functionality from the parent

class. It is achieved with the help of the extends and implements keywords."

- **"Which are the different types of inheritance?"**
  - "Single, Multi-level, and Multiple inheritance."
- **"Is multiple inheritance supported by Java?"**
  - "Through classes, it is not supported, but through interfaces, we can achieve multiple inheritance."
- **"Why is it not supported through classes?"**
  - "Due to the ambiguity problem."
- **On Spring Boot Annotations**
- **"What is the difference between @GetMapping and @PostMapping?"**
  - "@GetMapping is used to get the data from the database, and @PostMapping is used to add the data into the database."
- **"Which are the other annotations that you have used in Spring Boot?"**
  - "@RestController, @PathVariable, @RequestBody, @Service, @Entity, and @Id."
- **"And what is the use of the @Table annotation?"**
  - "If we want to give a different name to our table apart from the entity class name, then we can use the @Table annotation."

17 LPA OFFER INTERVIEW

Of course! Here are the notes from the video, formatted in a way that is easy to copy for Word or use in Canva.

Advanced Java & Spring Boot: Interview Prep Guide

Topic 1: Core Java Concepts

Question: What is the difference between an Abstract Class and an Interface?

Answer:

- **Abstract Class:**
  - Uses the abstract keyword.
  - Can have both **abstract** (unimplemented) and **non-abstract** (implemented) methods.
  - **Has a constructor** to initialize the state of its objects.
  - Cannot be directly instantiated (you can't do new MyAbstractClass()).
- **Interface:**
  - Uses the interface keyword.
  - Before Java 8, it could **only** have abstract methods.

        - Does **not** have a constructor as it has no state to initialize.

Topic 2: Java 8 Features

Question: Why were Default and Static Methods added to interfaces in Java 8?

Answer:

- **Purpose:** To allow developers to **add new methods to existing interfaces** without breaking the classes that already implement them. It provides backward compatibility.

Question: What is a Functional Interface?

Answer:

- **Definition:** An interface that contains **exactly one abstract method**.
- **Use Case:** They are the foundation for using **lambda expressions** and method references, which leads to more readable and concise code.

Question: What is the difference between a Sequential Stream and a Parallel Stream?

Answer:

- **Sequential Stream:** Processes data in a single sequence, one element after another.

- **Parallel Stream:** Breaks the data into smaller chunks and processes them concurrently on multiple threads, which can offer better performance for large datasets.

Question: What does Immutability mean in Java?

Answer:

- **Definition:** An object is immutable if its **state cannot be modified** after it is created.
- **Behavior:** If you try to change an immutable object, you get a **new object** with the new value instead.
- **Examples:** The String class and all wrapper classes (Integer, Double, etc.) are immutable.

Topic 3: Java 17 Features

Question: What are Sealed Classes?

Answer:

- **Purpose:** A feature that allows you to **restrict which classes or interfaces can extend or implement them**.
- **Benefit:** Provides better **encapsulation** and gives the author more control over their code.

Question: What is the JIT (Just-In-Time) Compiler?

Answer:

- **Function:** It is a component of the JVM that improves performance by **compiling frequently used bytecode into native machine code** at runtime. This saves compilation time for recurring code blocks.

Topic 4: Spring Boot Concepts

Question: What is Aspect-Oriented Programming (AOP)?

Answer:

- **Purpose:** A programming paradigm that helps separate **cross-cutting concerns** (like logging, security, or transactions) from the main business logic.
- **Key Term - Advices:** These are the actions taken by an aspect at a specific point in the code (e.g., before a method runs, after it finishes).

Question: How are Transactions managed in Spring Boot?

Answer:

- **Annotation:** Using the @Transactional annotation.

- **Function:** It automatically handles the begin, commit, and rollback operations for your database transactions.
- **Propagation Levels:** Defines how a transaction relates to an existing one (e.g., REQUIRED uses an existing transaction, while REQUIRES_NEW always starts a new one).
- **Isolation Levels:** Defines how to protect the integrity of your data when multiple transactions are happening at once (e.g., SERIALIZABLE is the strictest level, locking the table to prevent conflicts).

Topic 5: Collections Framework

Question: What is the difference between a Vector and an ArrayList?

- **Answer:**
  - **Vector:** Is **thread-safe** (synchronized), which makes it slower.
  - **ArrayList:** Is **not thread-safe**, which makes it faster.

Question: What are Concurrent Collections (like ConcurrentHashMap)?

Answer:

- **Purpose:** To provide both **thread safety and better performance**.
- **Mechanism:** They use advanced techniques like segment-level locks, allowing multiple threads to operate on different parts of the collection at the same time without conflict.

Topic 6: JPA and Hibernate

Question: What is Hibernate?

Answer:

- **Definition:** It is an **ORM (Object-Relational Mapping)** tool.
- **Role in Spring:** It is the default **implementation** for Spring Data JPA, handling the connection to the database and the mapping between your Java objects and database tables.

INTERVIEW ASHUTOSH HVPM

.

Comprehensive Java & Spring Boot: Interview Q&A Guide

Topic 1: Introduction & Core Java

Question: Can you introduce yourself and your technical skills?

- **Answer:**
  - **Name:** Ashutosh
  - **Background:** Completed BCA in 2022.
  - **Skills:** Core Java (OOPs, Collections, Exception Handling), Spring, Spring Boot, Hibernate.
  - **Design Patterns:** Aware of **Factory**, **Template**, and **Singleton** patterns.

Question: What is Polymorphism?

Answer:

- **Definition:** The ability of a member (method or variable) in Java to have **one name and many forms**.
- **Types:**
  - **Compile-Time Polymorphism:** Achieved via method overloading.

        - **Run-Time Polymorphism:** Achieved via **method overriding**.
    - **Rules for Method Overriding:**
        - Method name and arguments must be the **same**.
        - Return type can be the same or **covariant** (a subtype of the parent's return type).
        - Access modifier can be the same or have an **increased scope** (e.g., from protected to public).

Question: Which methods of the Object class are you aware of?

Answer:

- equals(), hashCode(), toString(), and instanceof.
- **equals():** By default, it compares object references (==). To compare object *content*, you must **override** it.
- **hashCode():** A unique code from the JVM. If you override equals(), you **must** also override hashCode().
- **toString():** By default, it prints the class name and hash code. To print the object's *state*, you must **override** it.

Topic 2: Spring & Spring Boot

Question: What is Inversion of Control (IOC)?

Answer:

- **Concept:** The control of creating and managing objects is "inverted" from the developer to the Spring framework.
- **Mechanism:** Spring scans for classes with stereotype annotations (like @Component, @Service), creates objects for them, and stores them in the **IOC container**.

Question: What is Dependency Injection (DI)?

Answer:

- **Concept:** Spring automatically "injects" required objects (dependencies) into other objects.
- **Benefit:** It helps make the application **loosely coupled**.
- **Types:** Field, Constructor, Setter, and Lookup Method injection.
- **Constructor vs. Setter Injection:**
  - **Constructor Injection:** Used for **mandatory** dependencies.

        - **Setter Injection:** Used for **optional** dependencies.
    - **Why avoid Field Injection?** It uses Java Reflection internally, which can be slower and cause performance issues if used extensively.

Topic 3: The Collections Framework

Question: What is the difference between Collection and Collections?

Answer:

- **Collection:** Is an **interface** at the root of the framework (with sub-interfaces like List, Set, Queue).
- **Collections:** Is a **utility class** that contains static helper methods (like sort(), synchronizedList()).

Question: How would you sort a list of Employee objects by salary?

Answer:

- **Best Approach:** Use a Comparator. It allows for custom sorting logic without modifying the Employee class itself and supports multiple sorting orders.

- **Alternative:** Use the Comparable interface, but this requires modifying the Employee class and only allows for a single, natural sorting order.

Question: What is the difference between a List and a Set?

Answer:

Feature	List	Set
Duplicates	Allows duplicate elements	Does not allow duplicates
Null Values	Allows multiple nulls	Allows only one null
Order	Maintains insertion order	Generally unordered (except LinkedHashSet)
Access	Can access elements by index (get())	Must iterate to find an element

Question: What is the difference between ArrayList and Vector?

Answer:

- **Vector:** Is **synchronized** (thread-safe), which makes it slower.
- **ArrayList:** Is **not synchronized**, which makes it faster.

Question: What is ConcurrentModificationException and how do you solve it?

Answer:

- **Cause:** It occurs when you try to modify a collection (like an ArrayList) while you are iterating over it.
- **Solution:** Use **concurrent collections** (e.g., CopyOnWriteArrayList). They prevent this error by creating separate copies for reading and writing, ensuring the collection isn't modified during iteration.

Topic 4: Spring Data JPA

Question: How does Spring Data JPA work?

Answer:

- **Mechanism:** At runtime, Spring creates a **proxy class** that implements your repository interface (e.g., JpaRepository). This proxy class contains the actual logic to connect to the database and execute the queries.

Question: How do you write custom queries in JPA?

Answer:

- **Annotation:** Use the @Query annotation.
- **Types of Queries:**
  - **JPQL (Java Persistence Query Language):** Database-independent. It's slower because it needs to be converted to a native query at runtime, but it's more flexible if you might change databases.
  - **Native Query:** Database-specific SQL. It's generally faster but ties your code to a specific database vendor.

Topic 5: HashMap Internals

Question: How does a HashMap work internally?

Answer:

- **put(key, value):**
  1. It calculates the hashCode() of the key.
  2. It uses this hash code to find a "bucket" (an index in its internal array).
  3. If the bucket is empty, it stores the new entry.
  4. **If the bucket is not empty (a hash collision):** It uses the equals() method to check if the new key is the same as an existing key in that bucket.

        - If equals() is **true**, it updates the value of the existing entry.
        - If equals() is **false**, it adds the new entry to the same bucket (usually as a node in a linked list or a tree).

Question: Why must you override hashCode() and equals() for custom classes used as keys in a HashMap?

- **Answer:**
    - **Rule:** If two objects are considered equal according to your equals() method, they **must** return the same hashCode().
    - **If you don't override them:** The default methods from the Object class will be used. This means even two Employee objects with the exact same data will have different hash codes and will not be considered equal, leading to incorrect behavior in the HashMap.

5 YEARS EXPERIENCE INTERVIEW

Of course! Here are the notes from the Java full stack interview with Karan, formatted for easy use in Word or Canva.

Advanced Java Full Stack: Interview Q&A Guide

Topic 1: Core Java & OOPs

- **Question: Can you explain Polymorphism?**
- **Answer:**
    - Definition: "Many forms." It allows a single action to be performed in different ways.
    - Types:
        - Method Overloading (Compile-Time): Same method name, different parameters, within the same class.
        - Method Overriding (Run-Time): A child class provides a specific implementation for a method that is already defined in its parent class.
    - Rules for Overriding:

- Method name and parameters must be the same.
- Access modifier scope can be increased (e.g., protected to public).
- Return type can be the same or covariant.

Topic 2: The Collections Framework

Question: What is the difference between ArrayList and Vector?

Answer:

- ArrayList: Not synchronized (not thread-safe), making it faster.
- Vector: Is synchronized (thread-safe), which introduces a performance overhead.

Question: Why use Concurrent Collections (like CopyOnWriteArrayList)?

- Answer:
  - Purpose: To achieve both thread safety and high performance.

- Mechanism: They use advanced locking mechanisms (like "bucket-level locks" in ConcurrentHashMap) that allow multiple threads to access different parts of the collection simultaneously, which is more efficient than the single lock used by Vector.

Question: What causes a ConcurrentModificationException?

Answer:

- Cause: It happens when you try to modify a collection (e.g., add or remove elements) while you are iterating over it using a standard iterator.
- Solution: Use concurrent collections, which manage separate copies for reading and writing to avoid this conflict.

Question: How do you sort collections with custom logic?

Answer:

- Comparable Interface:
  - Used for a single, natural sorting order.

        - Requires implementing the compareTo() method inside the class you want to sort.
        - Tightly coupled.
    - Comparator Interface:
        - Used for multiple, custom sorting orders.
        - Implemented in a separate class.
        - Loosely coupled and more flexible.

Topic 3: Design Patterns

Question: Explain the Singleton Design Pattern.

Answer:

- Purpose: To ensure that a class has only one instance and to provide a global point of access to it.
- Implementation:
    1. A private constructor to prevent direct instantiation.
    2. A private static variable to hold the single instance.

3. A public static getInstance() factory method to return the instance.
    - Thread Safety: The getInstance() method must be synchronized (or use a synchronized block with double-check locking) to prevent multiple threads from creating multiple instances simultaneously.

Topic 4: Multi-threading

Question: What is a better alternative to the synchronized keyword for thread safety?

Answer:

- ReentrantLock: It provides more flexibility than synchronized. It allows you to specify a waiting period for a thread to acquire a lock, which helps prevent deadlocks and offers fairness policies.

Question: What are the different ways to create a thread?

Answer:

1. Extend the Thread class.

2. Implement the Runnable interface (preferred, as it allows for multiple inheritance).
3. Implement the Callable interface (used when you need a return value from the thread).
4. Use Lambda expressions (since Java 8).

Question: How do you manage threads in an application?

- Answer:
  - ExecutorService: It is a framework for managing thread pools, which allows you to create and control threads efficiently.
  - Implementations: FixedThreadPool, CachedThreadPool, ScheduledThreadPool, etc.

Topic 5: Hibernate & Caching

Question: What is the difference between First-Level and Second-Level Cache in Hibernate?

Answer:

- First-Level Cache:

    - Scope: Associated with a single Hibernate Session.
    - Lifespan: Data is cached only for the duration of that session. It is destroyed when the session is closed.

Second-Level Cache:

    - Scope: Shared across multiple sessions.
    - Lifespan: Data persists even after sessions are closed, improving performance for frequently accessed data across the entire application.

Topic 6: Application Performance & Spring Security

Question: How can you improve application performance?

- Answer:
    - Caching:
        - In-Memory Cache (e.g., Caffeine): Fast, but data is lost if the application restarts.

        - Distributed Cache (e.g., Redis): Slower, but data persists as it's external to the application.
    - Database Indexing: Speeds up query retrieval.
    - Asynchronous Communication: Using message queues to decouple processes.

Question: Explain the flow of Session-based Security in Spring.

- Answer:
    1. A user's first request is intercepted by the Security Filter.
    2. The user logs in, creating an Authentication Object.
    3. The Authentication Manager validates the credentials against the database.
    4. If valid, the Authentication Provider confirms the authentication.
    5. A Security Context Holder is created, and a Session ID is generated and sent to the user.

6. For all future requests, the user sends the Session ID to remain authenticated.

- **Question: What is the main disadvantage of session-based security?**
- **Answer:**
    - Scalability: It is not scalable for a large number of users. Managing thousands of session objects in server memory is not feasible. This is why modern applications often use stateless alternatives like JWT (JSON Web Tokens).