Java Core

1. What is the SOLID principal?

   Single Responsibility Principle: Each class should have only one responsibility.

   Open-Closed Principle: Software entities should be open for extension but closed for modification.

   Liskov Substitution Principle: Subtypes must be substitutable for their base types.

   Interface segregation Principle: Clients should not be forced to depend on interfaces they do not use.

   Dependency Inversion Principle: High-level modules should not depend on low-level modules. Both should depend on abstractions.

2. What is OOP? Explain Inheritance, Encapsulation, Polymorphism, Abstraction with examples.

   OOP stands for Object Oriented Programming.

- Inheritance: A child class can inherit fields and methods from its parent class except those private ones.
- Encapsulation: data hiding + abstraction
- Polymorphism:
  1) Compile time: Method overloading. Methods in same class with same name but different parameters.
  2) Runtime: Method riding. Method name and parameters are the same in the superclass and the child class
- Abstraction: hiding details of implementation of a method, only expose the method to get the information

3. What is method overriding and method overloading? Provide your own example.

   Compile time: Method overloading. Method name and parameters are the same in the superclass and the child class

   Runtime: Method riding. Methods in same class with same name but different parameters

4. Differences between interface and abstract class.
   1) A class can only extend one abstract class, but it can implement multiple interfaces
   2) An abstract class can have a constructor, but an interface doesn't need it
   3) Type of Variables: Interface can only have static final variables, abstract class can have instance variables, static variables, and final variables
   4) Type of methods: Interface can have static, abstract, and default methods. Abstract class cannot have default method, but all concrete methods are inherently default
   5) Implementation: Abstract class can provide the implementation of the interface. Interface can't provide the implementation of an abstract class.

5. What is the diamond problem? How can we solve the diamond problem?

   It’s an ambiguity that can arise as a consequence of allowing multiple inheritance. We can use aggregation to solve the problem. For example, instead of A extends B and C, we put B and C as the instance variables in class D

6. Difference between Array and ArrayList
   1) Array size is fixed, but ArrayList size is dynamic
   2) The memory usage for Array is fixed upon creation. So it’s more efficient for accessing elements in the array, but can also lead to wasted memory if the array is not fully utilized.
   3) The type an array can store is fixed upon creation, but type of the ArrayList is determined when it is declared using a generic type parameter

7. Difference between ArrayList and LinkedList
   1) The get method for ArrayList is O(1), while LinkedList is O(n)
   2) The insert or delete method for ArrayList is O(n), while LinkedList is O(1)

8. What is an immutable class? How to create an immutable class?

   An immutable class is a class whose instances cannot be modified after they are created.

   Use the final keyword to create an immutable class.

9. Why is String immutable?

   String is immutable because of 4 reasons.
   1) Security: More secure since no one can modify the contents of the string
   2) Thread-safety: Multiple threads can safely access and manipulate strings without memory inconsistency issues
   3) Memory optimization: String objects are cached in the String pool. If a string with a specific value already exists in the pool, Java can reuse the same object instead of creating a new one.
   4) Performance: Java can reuse substrings without creating new objects, and perform concatenation using a StringBuilder instead of creating a new string object each time.

10. Explain the String pool.

    The string pool is maintained by the Java Virtual Machine (JVM). String Pool is a collection of Strings that are stored in the Java heap memory. When creating new String objects, Java returns the existing string if String pool already has a string with same value.

11. Difference between == and equals()

    == checks for reference equality, while equals() checks for value equality.

12. How does HashMap work internally?

When an element is added to the HashMap, a hashcode is calculated based on the key value, and the element is stored in the array at the index corresponding to the hash code. If two different keys have the same hash code, a collision occurs, and the elements with the colliding keys are stored in a linked list at that array index. This linked list is called a bucket. When retrieving an element from the HashMap, the hash code of the key is used to locate the correct bucket. If there are multiple elements in the bucket, the equals() method is used to search the linked list for the element with the matching key.

13. How does HashSet work internally?

HashSet is implemented using a HashMap. When an element is added to the HashSet, it's actually added as a key to the HashMap with a dummy value "PRESENT". If there is already an element in the same bucket with the same hash code, the equals() method is used to determine whether the two elements are equal or not. If they are equal, the new element is not added to the HashSet since it already exists. If they are not equal, the new element is added to the bucket as a separate key.

14. Difference between HashMap, SynchronizedMap, and ConcurrentHashMap

HashMap is not thread safe. SynchronizedMap is thread safe but not as efficient. ConcurrentHashMap is both thread safe and efficient.

15. Difference between List, Set, Map, Queue

List is an ordered collection that allows duplicates, Set is an unordered collection of unique elements, Map is a collection of key-value pairs, and Queue is a collection of elements that supports insertion and removal at opposite ends.

16. What are types of exceptions?

1) Checked exceptions: caught by compiler during compile time. ClassNotFoundException, IOException
2) Unchecked exceptions: caught by JVM during runtime. NullPointerException, ArrayIndexOutOfBoundsException

17. How to create customized exceptions?

1) Create a new Java class that extends the Exception class or one of its subclasses
2) Define a constructor for your custom exception class that takes a message as a parameter and passes it to the superclass constructor using the super() keyword
3) Use the custom exception by throwing it in your code when the desired situation arises

18. How do you handle exceptions in Java?

Use try and catch block, use ExceptionHandler, or simply propagate the exception up by adding the throws clause to the method signature

19. How are exceptions in your web application handled?

I used ExceptionHandler to handle exceptions. It separates the code that handles exceptions from the code that throws exceptions. This separation of concerns makes the code more modular and easier to maintain.

20. Explain about Exception Propagation.

When an exception is thrown by a method, it is propagated up the call stack until it is caught by an appropriate exception handler or it reaches the top of the call stack.

21. Difference between final vs finally vs finalize.

1) Final is a keyword used to mark a variable, method or class as constant
2) Finally is used in a try-catch-finally block to specify a block of code that will always be executed, regardless of whether an exception is thrown or not.
3) Finalize is a method in the Object class that is called by the garbage collector when an object is about to be garbage collected.

22. How to create a thread?

1) Extend the Thread class. Execute it by run()
2) Implement a Runnable class. Execute it by passing the Runnable to a thread and start()
3) Use lambda expression to implement the run() method of a Thread class.
4) Use ExecutorService with Callable or Runnable

23. Difference between Runnable and Callable.

Runnable won't return anything and does not throw any exception.

Callable will return a Future object which can be used to retrieve the actual result later and can throw a checked exception

24. Why wait() and notify() are in Object class but not in Thread Class.

wait() and notify() are methods in the Object class because they are used to synchronize access to shared objects between threads. These methods are used to coordinate threads so that they do not access shared resources simultaneously.

25. What is join()?

join() allows one thread to wait for the completion of another

26. Explain thread life cycle.

1) New: When a thread is created but has not yet started, it is in the new state. In this state, the thread is considered alive but not yet running.
2) Runnable: Once the start() method is called on a thread, it enters the runnable state. In this state, the thread is ready to run, but it may not actually be running because the scheduler has not yet selected it for execution.
3) Running: When the scheduler selects a thread for execution, it enters the running state. In this state, the thread is actually executing its task.
4) Blocked/Waiting/Sleeping: A running thread can enter a blocked state for several reasons. For example, if it is waiting for I/O to complete or for a lock to be released, it may enter a blocked state. A thread can also enter a waiting or sleeping state if it is explicitly programmed to do so using methods like wait(), sleep(), or join().
5) Terminated/Dead: A thread can reach the end of its task and terminate, or it can be terminated prematurely using the stop() method. In either case, the thread enters the terminated state.

27. Difference between sleep() and wait()

1) sleep() belongs to Thread and wait() belongs to Object.
2) sleep() suspends execution for a specified period. While wait() forces the current thread to wait until some other thread invokes the notify() or notifyAll() method on the same object.

28. Difference between Future and CompletableFuture.

They are both used for Async programming.

Future is a placeholder object that allows the code to continue executing and later retrieve the result of the computation when it becomes available.

CompletableFuture provides additional methods to work with the result. For example, it allows chaining of multiple tasks and callback functions to be executed when the task is completed

29. What is the advantage of using executor service instead of simply creating a thread and running it.

ExecutorService maintains a pool of threads and assigns them tasks. It allows for efficient use of system resources by managing thread creation and re-use, which can result in improved performance and scalability.

30. What is synchronization? How do you do synchronization?

Synchronization is to control access to shared resources to prevent race conditions and ensure thread-safety. We can do synchronization by using the synchronized keyword in the method signature or by using the synchronized keyword around any block of code

31. What is a marker interface? Provide an example and explain how it works.

    A marker interface is an interface that has no methods or constants inside it. It provides run-time type information about objects, so the compiler and JVM have additional information about the object. For example, Serializable is a marker interface used to mark a class as being serializable, meaning that objects of that class can be converted into a stream of bytes and then reconstructed later.

32. What does Serialization mean?

    It means the objects of that class can be converted into a stream of bytes and then reconstructed later.

33. What's the purpose of transient variables?

    The transient keyword is used to mark a variable as non-serializable. Any variables marked as transient are excluded from the serialization process.

34. How do you do the serialization and deserialization?

    We can use ObjectOutputStream to convert an object into a stream of bytes.

    We can use ObjectInputStream to convert a stream of bytes back into an object.

JUnit

1. Which version of JUnit are you using?
   JUnit 5

2. What is the difference between JUnit 4 and JUnit 5?
   a. JUnit 5 supports more annotations like: @BeforeEach, @AfterEach
   b. JUnit 5 provides more kinds of assertions like: assertThrows, assertTimeout

3. Can you explain the three components of JUnit5 and explain about their responsibilities?
   a. JUnit Platform: foundation to launch the test framework
   b. JUnit Jupiter: test framework for writing and running tests in JUnit 5
   c. JUnit Vintage: provides backward compatibility with JUnit 3 and JUnit 4

4. What is DDT (Data Driven Testing) and why do we need it?

   Instead of hardcoding the test data into the test scripts, DDT separates the test data from the test logic, allowing testers to use external data sources such as spreadsheets, databases, or CSV files to provide input to the test scripts.

   good at: Reusability, Scalability, Maintainability, Flexibility, Coverage

5. In JUnit 5, what is the execution order of test methods inside a test class by default?

   JUnit 5 promotes independent tests, so they are run in parallel

6. In JUnit 5, how do you specify the execution order of test methods within a test class
   Use @TestMethodOrder and @Order annotation

7. What is test case isolation and how does JUnit 5 perform test case isolation?
   It means tests should be independent of each other. JUnit perform test case isolation by fresh instance, @BeforeEach, @AfterEach, assertions, parallel execution

8. In JUnit 5, which methods must be declared as static by default?
   @BeforeAll and @AfterAll

9. In JUnit 5, what is the default lifecycle of a test instance?
   Default is per method, but you can change it using @TestInstance

10. In JUnit 5, what will happen if one assertion fails? Will the remaining assertions in the same test method still be executed?
    It will be stopped and remaining assertions won't be reached

11. In JUnit 5, how to guarantee that all the assertions are executed regardless of assertions results
    Use assertAll

12. What is the difference between assertTimeout() and assertTimeoutPreemptively() methods in JUnit 5?
    assertTimeoutPreemptively returns false if timeout is reached

13. What is race condition and how to prevent it during parallel testing?
    Use synchronized keyword, use atomic number, or use thread-safe data structures

Selenium WebDriver

1. What is Selenium WebDriver?
   It's used to perform automated testing of web applications.

2. What should I do if I want to switch my browser from Chrome to Firefox during a UI Testing using Selenium WebDriver?
   Download firefox webdriver and instantiate firefox driver instead of chrome driver

3. What is the difference between get() and navigate().to() in Selenium WebDriver?
   navigate().to() allows for additional navigation options such as refreshing the page, navigating forward or backward in the browser's history

4. What is implicit waiting in Selenium WebDriver? Why do we need this?
   It makes WebDriver to wait for a certain amount of time before throwing a NoSuchElementException if an element is not immediately found in the DOM

5. Can you explain different kinds of locators in WebDriver?
   id, name class, xpath, css, link, tag

6. How do you select a specific option in a dropdown selectors using Selenium WebDriver? (At least two ways)
   Locate the select element, then select by id or value

7. How do you handle an alert window that has cancel, accept and input text boxes?
   driver.switchTo().alert(), then use dismiss, accept, or send key

8. How do you switch between different windows or tabs in Selenium WebDriver?
   driver.getWindowHandles();

9. How do you upload a file using a file upload button in WebDriver?
   Find the input element and use fileInput.sendKeys(filePath). Then find the button and click

10. In Selenium WebDriver, how do you store elements of a web table into a 2D array?
    Get the table element, get the tr element. For each tr, loop through all table data and store them in a 2D array

11. In Selenium, how do you perform drag and drop operations?
    Locate the drag element and drop element, use actions.dragAndDrop(sourceElement, targetElement).build().perform();

12. What is Actions class in Selenium? Tell me about some actions you can perform in Selenium.
    Actions class provides a way to perform user interactions such as mouse movement or key press. For example, double click, right click, or key combinations

13. How do you scroll up and down the webpages in Selenium?
    Need to use JavascriptExecutor. js.executeScript("window.scrollBy(0,-1000)");

14. Why can we convert an instance of class “WebDriver” to the class of “JavascriptExecutor”?
    Because WebDrivers like ChromeDriver also implements JavascriptExecutor

15. Can you explain different kinds of Waits in Selenium WebDriver?
    Implicit waits are used to set a default waiting time, Explicit waits are used to wait for a certain condition to occur

16. How do you run your Selenium WebDriver code in a testing environment without Graphic Interfaces?
    You could enable headless mode by adding headless option

17. What are dynamic web elements and how do you handle dynamic web elements in Selenium WebDriver?
    Using Stable Attributes, Use explicit or implicit waits, Dynamic XPath Selectors

18. Can you explain necessary maven dependencies in a UI Testing framework?
    Selenium WebDriver and the specific driver like ChromeDriver

19. Can you explain the process of building a UI Testing framework?
    Use a build tool like maven, add all dependencies. Create project structure, add packages for test cases, configurations, and page objects. Write the test cases. Integrate with CI/CD

20. How to write a feature file using cucumber, can you give me an example?
    BDD software development methodology that focuses on collaboration among stakeholders to define and understand the behavior of a system through examples written in natural language. cucumber is a BDD tool written in this pattern: scenario, given, when, and, then, and

21. What is Page Object Modeling?
    POM is a design pattern used in test automation to enhance the usability and readability of test scripts. Each web page in an application is represented by a separate class, known as a Page Object. These Page Objects encapsulate the locators and methods needed to interact with the elements on the respective web page.

22. .How to initialize all the web elements in a page object model annotated with @FindBy?
    In page constructor, use PageFactory.initElements(driver, this);

23. Tell me about the four OOP principles you applied in your UI Testing framework.
    Inheritance: create a hierarchy of Page Objects
    Polymorphism: override different methods
    Abstraction:
    Encapsulation:

24. Tell me about the design patterns you applied in your UI Testing framework
    Page Object Model, Singleton Pattern, Factory Method Pattern

Java 8

1. Have you used Java 8? What features are you familiar with?
    1) Lambda expressions: A concise way to express functionality as an expression, allowing for functional programming paradigms to be used in Java.
    2) Stream API: allows easy and efficient processing of data
    3) Default methods: A way to add methods to interfaces without breaking backward compatibility.
    4) Functional interface: predicate, consumer, supplier
    5) Optional: A container object that may or may not contain a non-null value, providing a way to avoid null pointer exceptions.
    6) Date and time API: A new API for handling dates and times in Java, replacing the older java.util.Date and java.util.Calendar classes.

2. What is the lambda expression?

   Lambda expression is a concise way to define a functionality without writing an actual method or anonymous inner class

3. What is the functional Interface? a. Follow Up: Is @FunctionalInterface annotation required?

   A functional interface in Java is an interface that defines a single abstract method.

   It's not necessary. But it's recommended because it will throw compilation error if more than one abstract method is added in the future.

4. How do lambda expressions and functional interfaces work together? Provide your own example, implementing a functional interface using lambda expression

   They work together to make the code concise.

```
@FunctionalInterface
interface MyFunctionalInterface {
   int myMethod(int a, int b);
}

class LambdaExample {
   public static void main(String[] args) {
       MyFunctionalInterface myFunction = (a, b) -> a + b;
```

```
        int result = myFunction.myMethod(5, 3);
        System.out.println("Result: " + result); // Output: Result: 8
    }
}
```

5. Comparator vs Comparable
   Both are used for custom ordering. To use Comparable, we have to override the compareTo method first. To use Comparator, we just need to implement it using a lambda expression.

6. What is stream API in Java 8?
   Stream API allows easy and efficient processing of data.
7. What is terminal operation and what is intermediate operation? Name some operations that you used.
   Intermediate operations: map(), filter, limit(), sorted()
   Terminal operations: forEach(), collect(), anyMatch(), findAny()

8. Differences between stream and collection.
   Stream doesn't modify the data. It just applies a set of procedures to the data to produce the desired result. Collection is an in-memory data structure, which holds all the values that the data structure currently has

9. Is stream API sequential or parallel? How do we do parallel streams?
   Stream API is sequential by default. We can use parallelStream() to do parallel streams.

10. What is flatMap?
    flatMap allows you to transform each element in a stream into another stream and merge all of the resulting streams into a single stream.

11. What is the default method and what is the static method?
    Default method provides a default implementation for a method in the interface, which can be overridden by any implementing class. Static method provides a utility method that can be called directly on the interface without the need for an instance. Useful for Util classes.

12. What is Optional in Java 8?
    Optional is a container that may or may not contain a value. It can be used to circumvent handling NullPointerExceptions.

Design Patterns

1. What design patterns have you used?
   Singleton Pattern, builder design pattern, and Factory Pattern

2. What is the Singleton Design Pattern?
   Singleton Pattern: This pattern ensures that only one instance of a class is created and provides a global point of access to that instance

3. How to create a singleton? (eager initialization and lazy initialization) Please provide an implementation.

   To create a singleton, you need to ensure that only one instance of a class is created and provide a global point of access to that instance.

   Eager:

```java
public class Singleton {
   private static final Singleton INSTANCE = new Singleton();

   private Singleton() {}

   public static synchronized Singleton getInstance() {
       return INSTANCE;
   }
}
```

   Lazy:

```java
public class Singleton {
   private static Singleton INSTANCE;

   private Singleton() {}

   public static synchronized Singleton getInstance() {
       if(INSTANCE == null) {
           INSTANCE = new Singleton();
       }
       return INSTANCE;
```

```
    }
}
```

4. Is Singleton thread safe?

   No. Multiple threads can access the singleton same time and create multiple objects, violating the singleton concept

5. How to make singleton thread safe? (for both eager and lazy)

   For eager initialization, you can make the singleton thread-safe by initializing the instance variable statically and marking the constructor as private. This way, the instance is created only once, and no other instances can be created from outside the class.

   For lazy initialization, you can make the singleton thread-safe by using synchronization or other techniques to ensure that only one thread can access the instance variable at a time.

6. How to prevent Singleton Pattern from Reflection, Serialization and Cloning?

   1) Reflection: throw an exception in the constructor if the instance already exist
   2) Serialization: implement the readResolve() method.
   3) Cloning: override clone() method and throw an exception

7. What is the factory design pattern? Why do we use factories? How do you use this design pattern in your application?

   Factory design pattern is an object creation pattern that determines what will be created based on the type of object that the user wanted. We use factories to separate the process of object creation from the rest of the application code, increase security, and reduce complexity. In JDBC, the java.sql.DriverManager class provides a factory method getConnection() that returns a Connection object according to the database system we are using.

8. Provide a code example for factory design pattern

```
public interface Weapon {
    void attack();
}

public class Sword implements Weapon {
    public void attack() {
        System.out.println("Swish! The sword slices through the
air.");
    }
}
```

```java
public class Hammer implements Weapon {
    public void attack() {
        System.out.println("Thunk! The hammer crushes its target.");
    }
}

public interface WeaponFactory {
    Weapon createWeapon();
}

public class SwordFactory implements WeaponFactory {
    public Weapon createWeapon() {
        return new Sword();
    }
}

public class HammerFactory implements WeaponFactory {
    public Weapon createWeapon() {
        return new Hammer();
    }
}

WeaponFactory swordFactory = new SwordFactory();
Weapon sword = swordFactory.createWeapon();
sword.attack(); // Output: Swish! The sword slices through the air.

WeaponFactory hammerFactory = new HammerFactory();
Weapon hammer = hammerFactory.createWeapon();
hammer.attack(); // Output: Thunk! The hammer crushes its target.
```

9. Difference between factory vs. abstract factory design pattern

   Factory: A factory that creates objects that derive from a particular base class.

   Abstract factory: A factory that creates other factories, and these factories in turn create objects derived from base classes.

10. Provide a code example for builder design pattern

```java
@Builder
public class ResponseStatus{

   private Boolean status;
   private String message;

}
```

```java
ResponseStatus.builder().status(true).message("example").build();
```

11. Explain the API gateway design pattern

    The API Gateway design pattern provides a single entry point for all client requests to a system. The primary purpose of the API Gateway is to expose a unified interface for multiple services, thereby simplifying the client-side code and abstracting the underlying system architecture.

12. Explain the circuit breaker design pattern

    The Circuit Breaker design pattern is a software design pattern used to improve the resiliency and fault-tolerance of distributed systems.

    1) Closed state: In this state, the circuit breaker allows normal operation of the system. Requests are passed through to the underlying system, and responses are returned to the caller.
    2) Open state: If the underlying system experiences a failure or fault, the circuit breaker switches to the open state. In this state, all requests are rejected, and the circuit breaker returns an error message to the caller. This prevents further requests from overloading the system and potentially causing more damage.
    3) Half-open state: After a specified period of time, the circuit breaker transitions to the half-open state. In this state, a limited number of requests are allowed to pass through to the underlying system to test if it has recovered. If these requests are successful, the circuit breaker returns to the closed state. If they fail, the circuit breaker returns to the open state.

13. Explain proxy design pattern

    Proxy is a wrapper class of another object in order to control access to it or to add additional functionality. It can intercept requests and perform additional operations before or after forwarding them to the real object.

14. Explain Observer design pattern

    It’s a software design pattern in which an object, named the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

15. Explain Chain of Responsibility design pattern

    It allows a set of objects to handle a request in a sequential manner. Each object in the chain will handle the request or pass it on to the next object in the chain until the request is fulfilled.

JDBC, Hibernate, Database

1. What is SQL Injection? How to solve it?

   It means attackers inject malicious SQL code into an application's database through user inputs. We can solve it by using preparedStatements.

2. Difference between Statement and PreparedStatement
    1) Statement has no parameters
    2) Statement not as efficient

3. What are JDBC statements? List the types of JDBC statements and their usage.
    1) Statement: A Statement object is used to send a simple SQL statement to the database with no parameters. Not as efficient as PreparedStatement because the same query will be compiled n times if n rows need to be inserted.
    2) PreparedStatement: It sends precompiled statements to the databases with or without parameters. More efficient because only the parameters will be sent to the database
    3) CallableStatement: It's used to call pre-written procedures from the database.

4. What is JdbcTemplate? And what are some of the advantages it has over standard JDBC?

   JdbcTemplate is a class provided by the Spring Framework that simplifies the use of JDBC by encapsulating the common database operations and exception handling. It provides a simple and consistent interface for performing database operations such as querying, updating, and deleting data.

5. How to handle transactions manually in JDBC?
   We usually perform Transaction Management in JDBC by the Connection object. Default mode is autocommit, which executes statements individually. We usually turn it off so that no SQL statement will be committed until an explicit is invoked conn.commit().

6. What is ORM and what are its benefits?
   It stands for Object Relational Mapping. It maps a table from a database to an object in java. It makes developers focus on the business logic instead of repetitive query languages.

7. What is Hibernate? How to configure Hibernate? Follow Up: If we want to migrate from MySQL to OracleDB, what is the one configuration property that we need to change other than the database url, driver, username and password?
   Hibernate is an ORM framework that allows developers to map Java classes to database tables, and vice versa. To configure Hibernate, we need to add things like url, username, password, dialect to application.yml file. Then retrieve the information in HibernateProperty class. Finally, in HibernateConfig, setup DataSource, PackagesToScan, and HibernateProperties.

8. What is Session and SessionFactory in Hibernate? Follow Up: Is session in hibernate thread safe?
    1) SessionFactory is a thread-safe, immutable cache of compiled mappings for a single database. It is created only once during the application startup and shared by all the threads of the application. It's responsible for creating and managing Hibernate sessions.
    2) Session is a non-thread-safe object representing a single unit of work with the database.

9. Difference between getCurrentSession vs. OpenSession

    The getCurrentSession() method is used to obtain the current Session associated with the transactional context. The openSession() method, on the other hand, is used to obtain a new Session object every time it's called.

10. Difference between get() vs. load()
    1) Get returns null if no object can be found using a given identifier, while load returns a proxy object, and it throws an exception if you try to get the object from the proxy.
    2) Get is eager loading(returns a fully initialized object), load is lazy loading (returns a proxy object)
    3) Get is slightly slower

11. Difference between update, merge, saveOrUpdate
    1) Update throws an exception if you pass in a transient entity and it doesn't conform to JPA. It returns void.
    2) Merge can also update a record based on the passed in object and it conforms to JPA. It returns the newly updated object.
    3) SaveOrUpdate calls save if Id is not present, and it calls update if Id is present. Also, it doesn't throw an exception if you pass in a transient entity

12. Why do we use flush(), clear(), and commit()?
    1) flush(): used to synchronize the state of the managed entities with the database. It forces any pending changes to be written to the database immediately, even if the transaction is not yet committed. This method is useful when you need to ensure that any changes made to the entities are immediately visible to other parts of the application or to other transactions.
    2) clear(): used to detach all managed entities from the current persistence context. This means that all pending changes to the entities are discarded, and they become detached from the Session. This method is useful when you want to release the memory occupied by the managed entities, or when you want to detach them from the current transaction and use them in a different transaction.
    3) commit(): used to commit the current transaction and write any pending changes to the database. It ends the current transaction and makes all the changes made to the entities persistent. This method is called implicitly when the transaction is committed or when the Session is closed.

13. How to do Many-to-Many mapping in hibernate

We can use @many-to-many annotation. We need to specify the “joinColumns”, “inverseJoinColumns”, and “mappedBy” correspondingly.

14. Give an example with @ManyToOne and @OneToMany

Order to OrderItem is @OneToMany. OrderItem to Order is @ManyToOne.

15. Fetching strategy in Hibernate

a. What is lazy fetching?

With lazy fetching, Hibernate only loads the primary entity when it is requested, and does not load its associated entities until they are accessed.

b. What is LazyInitializationException?

This exception occurs when an attempt is made to access a lazy-loaded object outside of an active Hibernate Session.

c. How to prevent LazyInitializationException?

Use eager fetching. Or keep the session open during the request-response window.

16. Why is Hibernate often preferred over JDBC?

1) Hibernate is an ORM framework. This makes the code easier to read, write, and maintain.
2) Hibernate allows applications to work with different databases without changing the code.
3) Hibernate provides a caching mechanism that can improve performance by reducing the number of database queries.
4) Hibernate provides transaction management, ensuring data consistency and integrity.

17. What is HQL and what is Criteria? What is type safe?

1) HQL is an object oriented representation of Hibernate Query that allows you to express database queries in terms of Hibernate entities.
2) Criteria is an approach for building database queries using a type-safe API.

```
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Order> cq = cb.createQuery(Order.class);
Root<Order> root = cq.from(Order.class);
```

3) Type safety is a feature of programming languages that ensures that the code is free of type-related errors at compile time. Criteria is used for this exact purpose.

18. Caching in Hibernate/What are first-level cache and second-level cache and how are they accessed? Follow up: If the second-level cache is enabled, how are the caches accessed (the order) when trying to fetch an entity?

1) First-level cache: Hibernate first level cache is associated with the Session object. It’s enabled by default and there is no way to disable it.
2) Second-level cache: Hibernate Second Level cache is disabled by default but we can enable it through configuration. It’s SessionFactory-scoped, meaning it is shared by all sessions created with the same session factory
3) Hibernate will first check the second-level cache

19. Explain ACID

1) Atomicity: either all successful or none.
2) Consistency: ensures bringing the database from one consistent state to another consistent state.
3) Isolation: ensures that a transaction is isolated from other transactions.
4) Durability: means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc

20. Given an Employee table with ID, Department, and Salary, write a SQL query for the following:
    a. Find the number of employees in each department.
       select count(*) from employees group by department;
    b. Get the highest salary per department group.
       select max(salary) from employees group by department;
    c. Find the employees who have the top salary in each department.
       ```
       select name from employees where salary in
           (select max(salary)
               from employees
               WHERE department_id = e.department_id
               group by Department)
       ```
    d. Find all employees with the 3rd highest salary.
       ```
       select * from employee where
          salary = (select salary from employee order by salary desc limit 1 offset 2)
       ```

21. Explain SQL vs NoSQL databases
    a. When do you choose to use SQL DB and when to use NoSQL DB?
       1. SQL databases use a relational data model, which organizes data into tables with rows and columns, and there could be relationships between different tables.
       2. NoSQL databases use a variety of data models like key-value pairs. NoSQL databases are designed for scalability, flexibility, and performance, and can handle large amounts of unstructured data.
       3. I would use SQL DB when the data has a clearly defined structure and requires strong data consistency. Otherwise, I would use NOSQL DB

22. What is inner join, left join, right join?
    1) Inner Join: returns only the rows from both tables that have matching values based on the join condition.
    2) Left Join: returns all the rows from the left table (Table 1) and matching rows from the right table (Table 2) based on the join condition.
    3) Right Join: returns all the rows from the right table (Table 2) and matching rows from the left table (Table 1) based on the join condition.

23. Explain CAP
    1) Consistency: ensures that all nodes in a distributed system see the same data at the same time. Consistency is important for systems that require high levels of data accuracy.
    2) Availability: ensures that every request gets a response, without guaranteeing that the response contains the most recent version of the data.
    3) Partition Tolerance: ensures that the system continues to operate even when there is a network partition or a communication failure between nodes. Partition Tolerance is important for systems that operate in distributed environments.
       a. Is MongoDB CP or AP?

MongoDB is typically considered to be an CP database.

24. What are the two locking types in databases?
   1) Optimistic locking: the database relies on a version number to check if the data has been modified since it was last read. If the version number is different, it just updates the local version number to the newest one and tries again?
   2) Pessimistic locking: the database acquires locks when reading or updating data to prevent other transactions from reading or updating the data until the lock is released.

Spring

1. Difference between the Spring framework and Spring Boot
   Spring boot has embedded Tomcat. Spring boot supports auto bean configuration and package scan.

2. What is dependency injection? Why do we need dependency injection? How does spring IOC container work?

   Dependency injection allows objects to depend on other objects or services without creating them directly.

   DI improves testability and it's easier to swap other pieces of code because the pieces are not dependent on one another

   Spring IOC container reads the configuration file and initializes the objects and their dependencies when application starts. It will then create instances of defined objects and inject it when needed.

3. How to inject beans in spring?
   You can inject beans by setter, constructor, or inject in the field directly.
   a. Follow up: Setter vs. constructor injection
      1) Constructor is prone to circular dependency because injections happen before object creation. Setter supports partial injections, it's a solution to circular dependency.
      2) Setter is preferred when properties are less and are mutable. Constructor is preferred when properties are more and are immutable.

   b. Follow up: How to make setter injection mandatory?
      Use the @Required annotation. This annotation ensures that the property must be set before the component can be used.

4. How to configure/create beans in spring?
    1) xml configuration, annotations like @Component and @Bean

5. Difference between BeanFactory and ApplicationContext?
    1) BeanFactory supports xml configuration only
    2) BeanFactory is lazy loaded, ApplicationContext is eager loaded

6. Describe bean scopes that are supported by spring

   Singleton, prototype, request, session, global session

    a. Singleton vs Prototype
        1) Singleton : only one shared instance of a singleton bean is managed
        2) Prototype: creation of a new bean instance every time a request is made
    b. Is singleton bean thread safe?

       No. Multiple threads can access the singleton same time and create multiple objects, violating the singleton concept

       or Yes, because there is only one instance of the object
    c. What's the default bean scope?

       singleton
    d. How to define the scope of a spring bean?

       @Scope("prototype")

7. Given a singleton bean A and a prototype bean B, how do we inject A into B? What will be the behaviors of the beans?

   We can inject A into B using one of the three injections. A will be created once during application startup, and a new B instance will be created when requested. If A changes, B will also change because all instances of B share the same instance of A

8. Given a singleton bean A and a prototype bean B, how do we inject B into A? What will be the behaviors of the beans?

   We can inject A into B using one of the three injections. A will be created once during application startup, and a new B instance will be created when requested. If B changes, A doesn't change because A has its only B instance

9. What’s your understanding on @Autowired? How does Spring know which bean to inject? What’s the usage of @Qualifier?、
   @Autowired is used to retrieve a bean from a Spring IoC container. It will check the type first. If there is only one type, it will inject it automatically.
   Sometimes we need to define multiple beans of the same type. In these cases, @Autowired will check @Qualifier and @Primary.
   If these are not setup, @Autowired will check the name.

10. What is circular dependency?
    Circular dependency happens when Bean A uses Bean B and Bean B uses Bean A.

11. What is the usage of @SpringBootApplication?
    Spring Boot SpringApplication class is used to bootstrap and launch a Spring application from a Java main method. It’s equivalent to @Configuration , @EnableAutoConfiguration and @ComponentScan

12. What's the difference between @Component, @Repository & @Service annotations in Spring?

All marks for classes that should be automatically scanned and registered as beans.

1) @Component: used to mark any Spring-managed component.
2) @Repository: used to mark classes that implement data access operations, such as DAOs
3) @Service: used to mark classes that provide business logic, like services
    a. Follow up: Can @Component, @Repository and @Service annotations be used interchangeably in Spring or do they provide any particular functionality besides acting as a notation device?
       No, they are identical in terms of functionality

13. How can you handle transactions in spring boot

The @Transactional annotation is used to mark a method or class as transactional, which means that any database operations performed within the method or class will be executed as part of a transaction.

14. Describe Spring MVC (Need to mention DispatcherServlet)

Spring MVC is the web framework built on the Servlet API that utilizes the MVC model for building web applications.

1) Request enters DispatcherServlet
2) Handler mapping choose the handler and handler adapter choose the controller
3) Controller execute the business logics and return the ModelAndView
4) Handler adapter get the ModelAndView and return it to DispatcherServlet
5) DispatcherServlet then use ViewResolver to resolve the view
6) DispatcherServlet render the view and return it to the user

15. What is ViewResolver in Spring?

The ViewResolver takes a logical view, such as JSP, and returns the actual views to DispatcherServlet.

16. Difference between @Controller and @RestController

@Controller is used to declare common web controllers which can return HTTP responses. @RestController is @Controller + @ResponseBody. It’s used to create controllers for REST APIs which can return JSON format response

17. What is RestTemplate?

RestTemplate is used to invoke one or more RESTful web services

18. @RequestBody vs. @ResponseBody

1) When a client sends a request with a JSON payload, the @RequestBody annotation can be used to map the request body to a Java object.
2) @ResponseBody annotation can be used to map the response body to a JSON

19. @PathVariable vs. @RequestParam

1) @PathVariable: extract variable from the url directly
2) @RequestParam: extract variable from the parameter payload

20. What are different kinds of http methods

Get, Post, Put, Patch, Delete

a. Follow up: Difference between POST vs PUT vs PATCH

   POST creates a resource. PUT update a resource. PATCH is similar to PUT. If you only need to update one field of the resource, you may want to use the PATCH method.

21. Difference between SOAP vs. REST

1) SOAP is a protocol while REST is an architectural style.
2) SOAP uses XML while REST can use multiple formats
3) SOAP requires more bandwidth and processing power compared to REST, because SOAP needs to have a header that contains routing data.
4) SOAP uses service interfaces to expose its functionality. REST use Uniform Service locators to access to the components

22. Is REST stateless?

Yes, meaning that calls are independent of one another, and each call contains all of the data necessary to complete itself successfully.

23. Describe the RESTful principles.

1) Client-Server: the server provides the required functionality to the client
2) Stateless: Client provides all info needed. Server doesn't store the info.
3) Cache: Recent request are saved in cache so that they can be retrieved real fast if the request is made again
4) Layered System: Each layer doesn't know anything about any layer other than that of immediate layer
5) Uniform Interface: It suggests that there should be a uniform way of interacting with a given server irrespective of device or type of application (website, mobile app)
6) Code on demand: Servers can also provide executable code to the client. For example, the server may provide executable JavaScript to the client

24. How to validate the values of a request body?

Use @Valid in front of the @RequestBody. Specify the constraint in the entity class.

a. How does BindingResult work?

BindingResult is used to capture and represent errors that occur during data binding between a web form and a domain object. If BindingResult.hasError(), you can print out the field errors.

25. How to maintain user logged in using REST service

Use a JWT token that contains user information like username.

26. What is Spring AOP?

Aspect oriented programming. Aspects enable the modularization of cross-cutting problems. For example, logging and exception handling.

27. Talk about Spring Security

Spring Security provides authentication, authorization, and other security features for Spring-based applications. It's built around the concept of security filters. These filters intercept incoming requests and perform various security-related tasks.

28. How is JWT used with Spring Security?

JWT is generated by the Authentication upon login based on a pre-defined key. It will be passed in to the controller as a bearer token. And the services have the same key to bind the token with a UserAuthDetails. An authorization object will be generated using the UserAuthDetails and it will be stored in SecurityContextHolder.

29. What is unit testing vs integration testing?

Unit testing tests the functionality of a single block of code without other dependencies. While integration testing tests code that have different dependencies

30. What do you use for testing? (Mockito)

I use Mockito and JUnit for testing.

31. Describe some common annotations of Mockito.

1) @ExtendWith(MockitoExtension.class): annotate the JUnit test with a MockitoExtension
2) @Mock: create and inject mocked instances
3) @Spy: create and inject partially mocked instances (parameterized constructor)
4) @InjectMocks: inject mock fields into the tested object automatically
5) verify: provide more ways to validate the business logic

a. How do you test a method that returns void?

If the void method alters the property of the class under test. Then you only need to check the value of the property.

32. What’s the difference between doReturn and thenReturn?
    1) thenReturn is used for mock object, it makes a real method call
    2) doReturn is used for spy object, it will not make a real method call

33. What are some tools that can be used to view test code coverage?

    Jacoco or IntelliJ

34. What annotation do you use to quickly switch between different environments to load different configurations?

    @ConfigurationProperties or @Profile

35. What is Jasypt?

    It’s one way to encrypt the database password in our properties file.

Microservices

1. Difference between Monolithic vs. Microservice. a. Advantages and disadvantages of Monoliths vs Microservices
    1) Monolithic: we put everything in one project. It's simple to develop, deploy, and scale.
    2) Microservice: allows individual services to be developed, deployed, and scaled independently. One service's failure does not affect the entire application. Different services can use different technologies. But it's more complicated to develop and deploy

2. What is cascading failure? How to prevent this failure?

    Cascading failure in microservices occurs when a failure in one service leads to a chain reaction of failures in other services that depend on it, ultimately causing the entire system to fail. We can use circuit breakers to prevent it.

3. What is fault tolerance? How to make your microservice fault tolerance?

    Fault tolerance is the ability of a system to continue to operate even when one or more components fail.
    1) deploy multiple instances of a microservice
    2) designing each microservice to be independent of one another
    3) use circuit breaker
    4) Monitoring and logging a microservice to collect valuable information for debugging and troubleshooting.

4. How do microservices communicate?

    Eureka provides a centralized directory where services can register their availability and location, allowing other services to discover and communicate with them easily.

5. How do you document your endpoints? What's the purpose of Swagger?

    We can use Swagger to document the endpoints. Swagger helps users build, document, test and consume RESTful web services.

6. How do you monitor your application?

    Use spring boot actuator. We can manage and monitor our applications by using HTTP endpoints. Auditing, health, and metrics gathering can also be automatically applied to your application

7. What is the gateway and is it necessary?

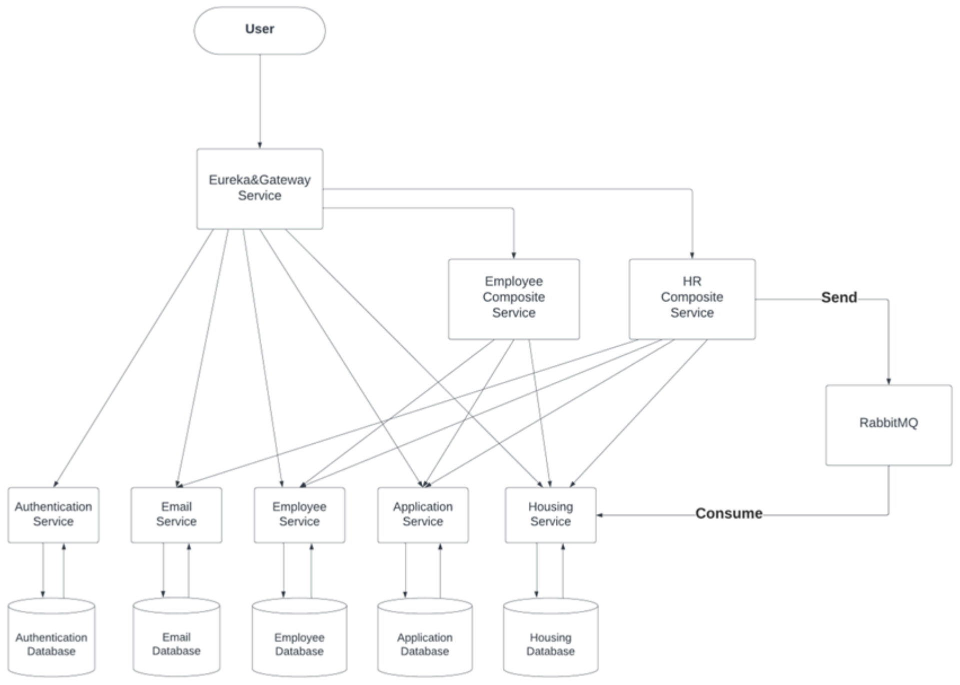
    It's not necessary, but it provides a centralized entry point for clients to access multiple microservices, making things easier to manage.

8. How many environments can your application have?
   Three environments: development, testing, and production

9. How did you deploy your application? (Answer: IntelliJ -> GitHub -> Github Actions/Jenkins -> Docker Hub -> AWS EC2/Kubernetes ->AWS EKS)
   1) Develop the application using IntelliJ and store the code in GitHub.
   2) Use a continuous integration tool like GitHub Actions or Jenkins to automate the build and test process of the application. These tools can also be used to trigger the deployment process when changes are pushed to the repository.
   3) Use Docker to containerize the application and store the Docker image in a container registry like Docker Hub.
   4) Deploy the application to an AWS EC2 instance or Kubernetes cluster. These tools enable scalable and reliable deployment of containerized applications.
   5) If using Kubernetes, we can deploy the application to AWS EKS (Elastic Kubernetes Service), a managed Kubernetes service that simplifies deployment, scaling, and management of containerized applications on AWS

10. Usage of Jenkins. (Please refer to the CI/CD guide on the training portal under day 35)
    Jenkins is used to construct CI/CD pipelines. It can also automate the process of running different tests and generate reports and analysis.

11. Describe your microservice architecture (candidates were asked to draw the architecture during the interview)



12. How to debug your microservice? (In other words, how to troubleshoot when there is an error in the microservice application?)

    1) Logging: Use logging to capture important events and errors that occur within your microservice. Centralized logging tools like Elasticsearch, Logstash, and Kibana (ELK) can help you collect and analyze logs from different sources, enabling you to identify issues quickly.
    2) Unit and Integration Testing: Implement unit and integration testing to detect issues before they are deployed to production. Automated testing can help identify issues early in the development process and reduce the risk of bugs or errors in production.
    3) We can also use debugging tools to help identify and fix issues in code. Tools like intellij can help me do this. And also sometimes I need to replicate the environment where the issue occurred to reproduce and debug the issue. And then collaborate with other team members to identify and troubleshoot issues in your microservice.

13. The production support team reports that your application is running slow as the customer base becomes large, what can you do to increase the performance of your application? (async, caching, pagination)

    1) Async: implement asynchronous processing for long-running tasks, such as file uploads, data processing, or sending emails. This can improve the responsiveness of the application and reduce the load on the server.
    2) Caching: Implement caching to store frequently accessed data in memory or on disk, reducing the number of database or API calls required. This can significantly improve the performance of the application, especially for read-heavy workloads.
    3) Pagination: Implement pagination to limit the number of records returned in a single API call. This can help reduce the amount of data that needs to be processed, improving the performance of the application.

14. What is RabbitMQ and what can it help us to achieve in a web application?

    RabbitMQ is a message broker software that enables different applications, services, and systems to communicate with each other by exchanging messages. It implements the Advanced Message Queuing Protocol (AMQP) and supports a wide range of programming languages and platforms.

15. What are the components of RabbitMQ? Describe the role of each component.

1) Producer: A producer is an application that generates messages and sends them to a RabbitMQ exchange. The producer does not know which queue the message will be sent to, only the exchange that it should be sent to.
2) Exchange: An exchange receives messages from producers and routes them to the appropriate queue based on the exchange type and routing key. RabbitMQ provides four exchange types: Direct, Fanout, Topic, and Headers.
3) Queue: A queue is a buffer that stores messages until they are consumed by a consumer application. RabbitMQ supports multiple queues that can be bound to an exchange.
4) Binding: A binding is a configuration that connects a queue to an exchange based on the routing key. It specifies the criteria used to route messages from the exchange to the queue.
5) Consumer: A consumer is an application that subscribes to a queue and receives messages from it. Consumers can acknowledge messages or reject them, causing them to be re-queued or sent to a dead letter queue.
6) Connection: A connection is a network connection between a producer or consumer application and RabbitMQ. It manages the session state and provides error handling and reconnection capabilities.
7) Virtual Host: A virtual host is a logical grouping of exchanges, queues, and bindings within RabbitMQ. It allows different applications to use the same RabbitMQ instance without interfering with each other.

16. What are different types of exchanges that exist in RabbitMQ? Explain each exchange.

1) Direct: It routes messages to queues based on a message's routing key.
2) Fanout: It routes messages to all queues that are bound to it
3) Topic: It routes messages to queues based on a pattern match between the message's routing key and the queue's binding key
4) Headers: It routes messages to queues based on the message's header values

17. What is a dead letter exchange (DLX)?

A Dead Letter Exchange (DLX) is an exchange in RabbitMQ that is used to handle messages that could not be delivered to their intended destination. When a message cannot be routed to a queue, it is sent to a dead letter exchange instead of being discarded. This provides a way to handle messages that cannot be processed, such as messages that exceed a certain size, messages with invalid content, or messages that are not consumed by a consumer within a certain timeframe.

18. How to secure your endpoint? (In other words, How can you check if a HTTP call is valid in microservices?)

1) Input Validation: Validate the input parameters of the HTTP call to ensure that they meet the expected format and criteria. This can include checking for required fields, validating data types, and sanitizing inputs to prevent injection attacks.
2) Authentication: Authenticate the caller of the HTTP call to ensure that they are authorized to access the microservice. This can include verifying credentials, checking for valid tokens or certificates, or implementing multi-factor authentication.
3) Authorization: Authorize the caller of the HTTP call to ensure that they have permission to perform the requested action on the microservice. This can include checking for roles, permissions, or attributes that allow the caller to access or modify the data or functionality of the microservice.
4) API Gateway: Use an API Gateway to manage and validate HTTP calls to your microservices. An API Gateway acts as a front-end to your microservices, providing a single entry point for HTTP calls and routing requests to the appropriate microservices based on rules and policies.
5) Implement Security Measures: Implement security measures such as rate-limiting, IP filtering, and monitoring to prevent attacks and detect anomalies.

19. Where do you store your configuration file when you use microservices?

1) Environment Variables: Microservices can be configured using environment variables that are set at runtime. This approach is useful for cloud-based deployments, where environment variables can be set through the cloud provider's management console or API.
2) Configuration Server: A configuration server can be used to store configuration files centrally. Microservices can then retrieve their configuration from the server at runtime. This approach allows for centralized management of configurations and can be useful for large deployments with many microservices.
3) Git Repository: Configuration files can be stored in a Git repository, and microservices can retrieve their configuration from the repository at runtime. This approach allows for version control of configurations and can be useful for teams that are already using Git for source code management.
4) Embedded Configuration: Configuration files can be embedded directly into the microservice code. This approach is useful for smaller deployments or for configurations that are unlikely to change frequently.

20. How did you do user authorization in microservices

I used JSON Web Tokens to provide secure and stateless authorization tokens that can be used to grant access to resources. Then, we could use role based or attribute based access control to authorize different users.

21. Vertical Scaling and Horizontal scaling in your application

1) Vertical Scaling: involves increasing the resources, such as CPU, RAM, or storage, of a single instance of a microservice. This involves upgrading the hardware or virtual machine that the microservice is running on. Vertical scaling can be relatively simple and fast, but it has limits to how much a single instance can be scaled.
2) Horizontal Scaling: involves increasing the number of instances of a microservice running in parallel to handle increased traffic and workload. This involves adding more instances of the microservice and load balancing traffic across them. Horizontal scaling can be more complex to set up, but it allows for more flexibility and scalability as traffic and workload increases.

22. Tell me about your experience with Cloud Service. Ex. AWS, GCP, Azure

23. What is Kafka? (Please look at the pdf and demo code on the training portal under day 35)

Kafka is used to build real-time streaming data pipelines. It is good at handling high-throughput data streams.

1) Producers are responsible for publishing data to Kafka topics, which are essentially named streams of data.
2) Consumers subscribe to these topics and read the data in real-time.
3) Brokers, which are essentially servers, act as intermediaries between producers and consumers. They are responsible for storing and distributing the data, as well as managing consumer groups, which are sets of consumers that work together to consume data from a topic.

24. What is ELK? (Please look at the guide on the training portal under day 35)

ELK is a popular open-source software stack used for log management and analysis. It consists of three main components:

1) Elasticsearch: Elasticsearch is a distributed, scalable search and analytics engine that provides fast search and powerful analytics capabilities. It is designed to store, search, and analyze large volumes of data in near real-time.
2) Logstash: Logstash is a data processing pipeline that ingests data from multiple sources, processes it, and sends it to Elasticsearch for storage and analysis. It can handle a wide range of input and output formats, making it easy to integrate with various data sources.
3) Kibana: Kibana is a data visualization tool that provides a user-friendly interface for querying and visualizing data stored in Elasticsearch. It allows users to create interactive dashboards and visualizations to explore and analyze their data.

25. Explain distributed database management (2-phase commit, SAGA)

1) 2PC is a protocol used for coordinating distributed transactions across multiple nodes. The protocol ensures that all nodes in a distributed transaction either commit or roll back the transaction together, ensuring data consistency across the distributed system.
    a) Phase 1 (Prepare phase): The coordinator node sends a prepare message to all participant nodes, asking them to vote whether they can commit or not. Each participant node replies with either "yes" or "no".
    b) Phase 2 (Commit phase): If all participant nodes vote "yes", the coordinator sends a commit message to all participant nodes, which then apply the changes to their local databases. If any participant votes "no", the coordinator sends an abort message to all participants, and the transaction is rolled back.

2) SAGA is an alternative approach to managing distributed transactions that allows for more flexibility and scalability. SAGA breaks a long-running transaction into smaller, more manageable, and atomic steps, with each step having its own transaction boundary. The individual steps of the transaction can be executed in any order and on any server, allowing for parallel processing and fault tolerance. Each step in the SAGA has a compensating action, which can be executed if a step fails, ensuring that the overall transaction can be rolled back if needed.