*Spring annotation*

*@Component Annotation :*

*Is class level annotation is denote class as component*

*Use := to make bean to manage spring component*

*Component responsible for some other annotation like*

## *@Service*

## *@ repository database*

## *@ Controller*

*org.springframework.stereotype*

*@Service*

1. *To indicate that hold as business logic*
2. *The utility classes can be marked as Service classes.*

*@ repository*

1. *to indicate that they’re dealing with CRUD operations, usually, it’s used with DAO (Data Access Object) or Repository implementations that deal with database tables*

*@ Controller*

1. *@Controller to indicate that they’re front controllers and responsible to handle user requests and return the appropriate response. It is mostly used with REST Web Services.*

***@Autowired Annotation***

1. *@Autowired annotation marks a Constructor, Setter method, Properties and Config() method as to be autowired that is ‘injecting beans'(Objects) at runtime by Spring Dependency*
2. *By declaring beans, you provide metadata to the Spring Container to return the required dependency object at runtime. This is called Spring Bean Autowiring*
3. *all the bean methods are defined in the class with****@configuration****annotation. At runtime, Spring will provide bean definitions by reading those methods.*

***@Entity***

*The Customer class is annotated with****@Entity****annotation and defines getters, setters, and constructors for the fields.*

## ***@Repository***

*@Repository is a Spring annotation that indicates that the decorated class is a repository. A repository is a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects. It is a specialization of the @Component annotation allowing for implementation classes to be autodetected through classpath scanning.*

***Difference Table***

|  |  |
| --- | --- |
| ***CrudRepository*** | ***JpaRepository*** |
| *It is a base interface and extends Repository Interface.* | *It extends PagingAndSortingRepository that extends CrudRepository.* |
| *It contains methods for CRUD operations. For example save(), saveAll(), findById(), findAll(), etc.* | *It contains the full API of CrudRepository and PagingAndSortingRepository. For example, it contains flush(), saveAndFlush(), saveAllAndFlush(), deleteInBatch(), etc along with the methods that are available in CrudRepository.* |
| *It doesn’t provide methods for implementing pagination and sorting* | *It provides all the methods for which are useful for implementing pagination.* |
| *It works as a marker interface.* | *It extends both CrudRepository and PagingAndSortingRepository.* |
| *To perform CRUD operations, define repository extending CrudRepository.* | *To perform CRUD as well as batch operations, define repository extends JpaRepository.* |
| ***Syntax:***  *public interface CrudRepository<T, ID> extends Repository<T, ID>* | ***Syntax:***  *public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>* |

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*@Requstmapping*

*Its use to map web request , its has many optional element like consumer , header ,method ,name params ,path produces and value we use class as well as method*

*Spring currently support five type of inbuilt annotation for handling incoming request which are GET ,POST ,DELETE PUT PATCH ,POST*

## Five types of Request in java

* *HTTP GET Request: This type of request is used to retrieve data from a server. It is commonly used to fetch resources or retrieve information from the server.*
* *HTTP POST Request: This type of request is used to send data to the server to create or update a resource. It is commonly used for form submissions, data uploads, or API requests that modify server-side data.*
* *HTTP PUT Request: This type of request is used to send data to the server to update or replace an existing resource. It is commonly used for updating or editing existing data on the server.*
* *HTTP DELETE Request: This type of request is used to delete a specified resource on the server. It is commonly used to remove data or resources from the server.*
* *HTTP PATCH Request: This type of request is used to send a partial update to a resource on the server. It is commonly used when only a portion of the resource needs to be modified.*

## What is json in java

*In Java, JSON (JavaScript Object Notation) is a popular format for exchanging and representing structured data. It provides libraries and APIs for parsing, generating, and manipulating JSON data . With JSON in Java, you can easily convert JSON to Java objects and vice versa, validate JSON against schemas, and integrate with JSON-based web services. It simplifies data interchange and enables seamless interaction with JSON-based systems.*

*JWT java*

*Steps :*

*Implement websecurityconfigadapter*

*override*

*🡪Configuration ()*

*🡪Configure()*

*Implement userdetialservise()*

*🡪userdetial();*

*🡪*

# *Java interview question*

## *paging in java in short*

# *what :*

# *paging refers to the process of dividing memory into fixed-size blocks called pages. However, Java abstracts away the low-level details of paging, as memory management is handled by the Java Virtual Machine (JVM).*

# *How:*

# *The JVM automatically manages memory through garbage collection and virtual memory techniques, so as a Java developer, you don't need to directly deal with paging. The JVM takes care of memory allocation and deallocation, allowing you to focus on writing your code.*

## *exception handling*

# *what*

*Exception handling is a programming concept that deals with handling and responding to unexpected or exceptional situations that occur during the execution of a program. When an exceptional situation, known as an exception, arises, it disrupts the normal flow of the program and can cause errors or undesired behavior.*

***How***

* *@ControllerAdvice: Spring Boot allows you to define a global exception handler using the @ControllerAdvice annotation. You can create a class annotated with @ControllerAdvice and define methods annotated with @ExceptionHandler to handle specific exceptions or groups of exceptions.*
* *@ExceptionHandler: By using the @ExceptionHandler annotation, you can define methods within your controllers to handle specific exceptions thrown by those controllers. These methods can return custom error messages, redirect to error pages, or perform any other necessary actions.*
* *Custom Exception Classes: You can create your custom exception classes by extending the base RuntimeException or Exception classes provided by Java. These custom exceptions can be thrown within your application's code, and you can handle them using the @ExceptionHandler methods mentioned earlier.*
* *@ResponseStatus: The @ResponseStatus annotation can be used in conjunction with custom exceptions to define the HTTP status code returned when the exception is thrown. By setting the appropriate status code, you can provide meaningful error responses to clients.*
* *Default Error Handling: Spring Boot automatically provides a default error handling mechanism that returns a standard error response when an unhandled exception occurs. This response includes details such as the error message, stack trace, and the HTTP status code.*

## Two type of exception in java

Checked Exceptions: *These are exceptions that are checked by the Java compiler at compile-time. Methods that can throw checked exceptions must declare them using the throws keyword. Examples of checked exceptions include IOException, SQLException, and FileNotFoundException. Checked exceptions represent exceptional conditions that can be reasonably expected and handled by the calling code.*

Unchecked Exceptions: *Also known as runtime exceptions, these exceptions do not need to be declared explicitly in the method signature. They occur at runtime and are not checked by the compiler. Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException. Unchecked exceptions typically indicate programming errors or unexpected conditions that should be fixed in the code.*

## *Why java is not 100% object oriented programing language*

## *What :*

*In an object-oriented programming language, everything is treated as an object, including basic data types.*

## *why*

*because of some reason java is not 100 % object-oriented because*

* *it includes primitive data types, such as int, float, and boolean, which are not objects and do not have the same capabilities as objects.*
* *it supports procedural programming constructs, such as static methods and variables, which do not rely on objects.*

## *Why pointers are not used in java*

## *What is pointer*

## *Pointers are variables that store memory addresses. They are used in low-level languages like C and C++ to directly manipulate memory and access data structures. Pointers allow for efficient memory management and indirect modification of data*

## *Why not allow*

*Because of some security reasons java did not support pointer*

* *They are unsafe*
* *Increases complex city and java is known for simplicity*
* *Since JVM is responsible for the implementation of memory allocation this avoids direct access to memory by the user*
* *Java references allow indirect manipulation of objects, similar to pointers, but they are strongly typed and automatically managed by garbage collection, eliminating manual memory management.*

## *JSON web token | JWT*

## *What :*

*A JSON web token(JWT) is JSON Object which is used to securely transfer information over the web(between two parties).*

*why:*

* *It can be used for an authentication system and can also be used for information exchange.*
* *The token is mainly Captain* ***header, payload, signature.*** *These three parts are separated by dots(.).*
* *JWT defines the structure of information we are sending from one party to the another, and it comes in two forms –****Serialized, Deserialized****.*
* ***Serialized*** *:The Serialized approach is mainly used to* transfer *the data through the network with each request and response.*
* ***Deserialized***

*While the deserialized approach is used to read and write data to the web token.*

*JWT in the deserialized form contains only the header and the payload. Both of them are plain JSON objects.*

Stream In Java

## *what*

* *In Java, a Stream is a sequence of elements that allows for functional and declarative processing of data collections.*
* *It provides operations like filtering, mapping, and reducing, and supports lazy evaluation and parallel processing. Streams enable concise and efficient manipulation of data in a more functional programming style.*
* *To create a stream, you can call the* ***stream()*** *method on a collection or array.*
* *Array to steam*

*int[] numbers = {1, 2, 3, 4, 5};*

*IntStream stream = Arrays.stream(numbers);*

* *Once you have a stream, you can perform operations on it such as filtering, mapping, and reducing. Here are a few examples:*

*// Filter out even numbers*

*IntStream filtered = stream.filter(n -> n % 2 == 1);*

*// Double each number*

*IntStream mapped = stream.map(n -> n \* 2);*

*// Compute the sum of all numbers*

*int sum = stream.reduce(0, (a, b) -> a + b);*

* *Note that these operations do not modify the original stream, but instead create a new stream with the modified elements.*
* ***Intermediate Operations:***

## *• map()*

*The map() method transforms each element of a stream into another object using a function that you provide. The function takes one input parameter and returns a transformed output.*

## *• Filter()*

*A filter in a stream can modify, transform, or extract data from the stream as it passes through the filter. Filters can be used for a variety of purposes, such as data validation, data transformation, data reduction, or data analysis.*

## *• Sorted()*

*The sorted method is used to sort the stream.*

*ascending*

*List<Integer> sortedNumbers = numbers.stream().sorted()*

*.collect(Collectors.toList());*

*descending*

*List<Integer> sortedNumbers =*

*numbers.stream().sorted((a, b) -> .compareTo(a)).collect(Collectors.toList());*

*Terminal Operations:*

## *• Collect()*

*When you apply a collect operation to a stream, it processes each element of the stream and adds it to the specified container. The resulting container contains all the elements of the original stream in a more structured format that can be easily manipulated or processed further.*

*forEach()*

*The forEach method is used to iterate through every element of the stream.*

*Spring boot basics \*\*\*\*\*\*\*\*\*\*\*\*\**

*The key concepts of Spring Boot include:*

* Auto-configuration: *Spring Boot automatically configures components based on dependencies and defaults.*
* Starter Dependencies*: Pre-configured dependencies that simplify dependency management for specific use cases.*
* Embedded Server: *An included server that allows running applications as standalone executables.*
* Actuator: *Provides endpoints and tools for monitoring and managing Spring Boot applications.*
* Spring Boot CLI*: Command-line tool for quickly creating and prototyping Spring Boot applications.*
* Configuration Properties*: Convenient way to configure applications using properties or YAML files.*
* Spring Boot Starters*: Simplify integration of third-party libraries or frameworks into Spring Boot applications.*
* Production-Ready Features*: Built-in features like metrics, health checks, logging, and security for production environments.*
* Spring Boot Actuator*: Provides endpoints and tools for monitoring and managing Spring Boot applications.*

## *Actuator:*

## *What :*

*provides production-ready features to monitor and manage applications.*

*It includes endpoints for health checks, metrics, logging, environment information, and more. Actuator enables better visibility and manageability of Spring Boot applications in production.*

*Spring Boot Annotations*

*: Spring Boot provides various annotations that simplify application development. These include @SpringBootApplication, which combines @Configuration, @EnableAutoConfiguration, and @ComponentScan; @RestController for creating RESTful endpoints; @Autowired for dependency injection; and more.*

*Externalized Configuration:*

*Spring Boot supports externalized configuration, allowing developers to configure applications using property files (application.properties or application.yml), environment variables, command-line arguments, or cloud configuration services. This promotes flexibility and portability.*

*Spring Boot Devtools: Spring Boot Devtools provides developer-focused tools, such as automatic restarts, live reload of static resources, and enhanced logging. It improves productivity by reducing development turnaround time.*

## *Rest and sope API*

* SOAP: Simple Object Access Protocol
* REST: Representational State Transfer
* *SOAP is a protocol, while REST is an architectural style.*
* *SOAP messages are typically larger and more complex, while REST messages are simpler and more lightweight.*
* *REST is stateless, whereas SOAP can have stateful interactions.*
* *REST APIs have a uniform interface using standard HTTP methods, while SOAP APIs have a standardized structure defined by the SOAP specification.*
* *REST APIs are generally easier to use and understand, while SOAP APIs can be more complex.*
* *REST APIs are often more scalable, while SOAP APIs may require more resources.*
* *REST has widespread adoption for public APIs and web applications, while SOAP is commonly used in enterprise and legacy systems.*

# *What are Microservices?*

* ***Microservice****is a small, loosely coupled distributed service.*
* *Microservice architecture evolved as a solution to the scalability, independently deployable, and innovation challenges with Monolithic architecture*
* *It allows you to take a large application and decompose or break it into easily manageable small components with narrowly defined responsibilities.*
* *It is considered the building block of modern applications. Microservices can be written in a variety of programming languages, and frameworks, and each service act as a mini-application on its own.*

## Reasons for Using Microservice

*In monolithic applications, there are a few challenges:*

* *For a large application, it is difficult to understand the complexity and make code changes fast and correctly, sometimes it becomes hard to manage the code*
* *Applications need extensive manual testing to ensure the impact of changes*
* *An application typically shares a common relational database to support the whole application*
* *For small changes, the whole application needs to be built and deployed*
* *The heavy application slows down start-up time*

## *Advantages of Microservices:*

* *Scalability: Microservices enable scaling individual components as needed, making it easier to handle high traffic and adapt to changing demands.*
* *Flexibility and Agility: Microservices allow for independent development, deployment, and updates, enabling faster iteration and continuous delivery. Different teams can work concurrently, accelerating development cycles.*
* *Fault Isolation: Each microservice operates independently, so a failure in one service doesn't bring down the entire system. This ensures system stability and resilience.*
* *Technology Diversity: Microservices support the use of different technologies for each service, promoting innovation and leveraging the strengths of various tools.*
* *Scalable Development: Microservices allow teams to work independently on specific services, leading to decentralized decision-making, increased productivity, and faster time-to-market.*

*Disadvantages of Microservices:*

* *Complexity: Microservices can be more complex than a monolithic system, making development, testing, and debugging harder.*
* *Operational Effort: Operating a microservices architecture requires extra effort to manage and maintain multiple services independently.*
* *Network Delay: Communication between microservices over the network can introduce delays, affecting system performance.*
* *Data Consistency: Ensuring consistent data across multiple services can be challenging due to decentralized data storage.*
* *Distributed System Challenges: Microservices introduce the complexities of distributed systems, such as service discovery, load balancing, fault tolerance, and inter-service communication. These challenges require robust infrastructure, monitoring, and management tools to ensure the smooth operation of the entire system.*

## How does a microservice differ from a monolithic architecture?

## monolithic

*a monolithic architecture is characterized by a single, tightly integrated application where all components are bundled together. It is typically large and complex, requiring the entire application to be redeployed for any changes. Scaling involves running multiple instances of the entire application, and a single technology stack is used throughout the system.*

## microservice

*microservice architecture is characterized by a collection of loosely coupled and independently deployable services. Each service focuses on a specific business capability and can be developed, deployed, and scaled independently. Microservices are smaller, allowing for faster and more frequent deployments. They offer granular scalability and flexibility in technology choices, as each service can use a different technology stack.*

# *ACID*

*Transaction protocols typically try to provide ACID guarantees, where ACID is an acronym for:*

* ***A****tomicity – All changes occur or none of the changes occur – prevents partial updates.*
* ***C****onsistency – The system as a whole move from one consistent state to another consistent state.*
* ***I****solation – Changes by one transaction aren’t seen by any other transaction until the transaction is complete - sometimes referred to as serializability meaning the results are the same whether transactions execute in parallel or are serialized.*
* ***D****urability – Once the outcome of the transaction has been determined, the outcome is durably recorded and will take place even in the presence of temporary failures.*

Microservices architecture is typically suitable in the following scenarios:

Complex and large-scale applications: Microservices are beneficial when dealing with complex applications that have multiple functionalities and require scalability. Breaking down the system into smaller, independent services makes it easier to manage and develop.

Independent deployment and scalability: If different parts of your application require frequent updates, scaling, or deployment independently, microservices can be advantageous. Each microservice can be developed, tested, deployed, and scaled independently, allowing for greater flexibility.

Polyglot development: Microservices architecture enables the use of different programming languages, frameworks, and technologies for each service. This is beneficial when different teams prefer different technologies or when specific services require specialized tools or libraries.

Team autonomy and agility: Microservices enable small, autonomous teams to develop and deploy their services independently. This promotes agility, as teams can work in parallel and release updates without coordination with other teams, reducing bottlenecks and enabling faster development cycles.

Scalability and fault isolation: Microservices facilitate horizontal scaling, allowing you to scale individual services based on demand. Additionally, if one microservice fails, it won't necessarily impact the entire system, as failures are isolated to specific services.

Rapid innovation and experimentation: Microservices architecture supports rapid innovation and experimentation by enabling the development of new features or services without disrupting the entire application. This allows for faster iterations and the ability to test and validate new ideas.

## What is the role of monitoring and logging in a microservice environment?

*In a microservice environment, monitoring and logging have the following key roles:*

* + *Monitoring: Collecting and analyzing metrics to ensure system health, performance, and availability.*
  + *Logging: Centralizing logs for troubleshooting, debugging, and auditing.*
  + *Distributed Tracing: Providing visibility into requests across multiple microservices for performance optimization.*
  + *Anomaly Detection and Alerting: Identifying abnormal behavior and triggering alerts for timely investigation.*
  + *Capacity Planning and Optimization: Forecasting resource requirements and optimizing capacity.*
  + *Security Monitoring: Detecting and responding to security events or suspicious activities*

## How do you ensure data security and privacy in microservices?

* *Use secure communication protocols like HTTPS/TLS.*
* *Implement authentication and authorization mechanisms.*
* *Employ Role-Based Access Control (RBAC).*
* *Utilize data encryption for storage and transmission.*
* *Validate and sanitize incoming data.*
* *Secure configuration management of sensitive information.*
* *Implement comprehensive logging and auditing.*
* *Mask or anonymize unnecessary sensitive data.*
* *Secure service-to-service communication.*
* *Conduct regular security testing and vulnerability assessments.*
* *Ensure compliance with data protection regulations.*

## How to communicate micrservicess?

* *HTTP/REST APIs for communication.*
* *Messaging/event-driven approach using message brokers.*
* *RPC frameworks for remote procedure calls.*
* *Service mesh frameworks for seamless communication and management.*
* *API gateways as a single entry point for routing requests.*
* *Shared libraries or SDKs to encapsulate communication logic.*

## *DevTools*

*Spring Boot DevTools is a module that enhances the developer experience during application development. It provides features like automatic restart, live reload, enhanced error page, database console, and additional development-time features. DevTools improves productivity, reduces development time, and simplifies common development tasks. It is meant for use in the development phase and is not recommended for production environments.*

## *What is a Spring Boot Actuator?*

*Spring Boot Actuator is a powerful feature of the Spring Boot framework that provides a set of production-ready tools and endpoints to monitor and manage Spring Boot applications. It enables developers and system administrators to gain insights into the application's health, metrics, environment, and more, allowing for efficient monitoring and troubleshooting in a production environment.*

*Key points to mention in an interview answer about Spring Boot Actuator:*

* *Purpose: The main purpose of Spring Boot Actuator is to provide operational and management features for Spring Boot applications in production environments.*
* *Key Features: Actuator offers a range of features, including health checks, metrics, environment information, auditing and tracing, management endpoints, security and access control.*
* *Health Checks: Actuator provides built-in health indicators to monitor the application's health status. Custom health checks can be added to assess specific components or dependencies.*
* *Metrics and Monitoring: Actuator exposes various metrics about the application, such as request rates, CPU usage, memory usage, etc. It integrates with popular monitoring systems for collecting and visualizing application metrics.*
* *Environment Information: Actuator provides detailed information about the application's environment, including configuration properties, system properties, and environment variables.*
* *Auditing and Tracing: Actuator includes endpoints for auditing and tracing requests, allowing tracking and monitoring of request details for analysis and debugging.*
* *Management Endpoints: Actuator offers a set of management endpoints that expose operational information and allow interaction with the application. It enables tasks like shutting down, restarting, or viewing thread dumps.*
* *Security and Access Control: Actuator supports securing management endpoints using Spring Security, allowing access control based on roles and permissions.*
* *Customization: Actuator provides extensibility and customization options, allowing developers to add custom health indicators, metrics, or management endpoints specific to their application's requirements.*
* *Integration: Actuator seamlessly integrates with other Spring Boot features and libraries, making it easy to incorporate into existing Spring Boot projects*

## *How can you access a value defined in the application? What is a properties file in Spring Boot?*

*Use the @Value annotation to access the properties which is defined in the application – properties file.*

*@Value("${custom.value}")*

*private String customVal;*

## *What is the primary difference between Spring and Spring Boot?*

* *Spring is a web application development framework based on Java.*
* *On the other hand Spring Boot is an extension of the spring framework which eliminated the boilerplate configuration required for setup a Spring application.*

## *Explain Spring Boot Admin*

* *Spring Boot admin is a community project which helps you to manage and monitor your Spring Boot applications.*

## *Can you disable the default web server in the Spring Boot application?*

*Yes, we can disable the default web server by using application.properties to configure the web application type.*

*spring.main.web-application-type=none*

## *What is @pathVariable?*

*@PathVariable annotation helps you to extract information from the URI directly.*

## *What is filter in spring boot*

*In a Spring Boot application, filters are components that intercept and process incoming requests and outgoing responses in a web application. Filters are part of the Servlet API and allow developers to perform pre-processing and post-processing tasks on HTTP requests and responses.*

*Key points to mention in an interview answer about filters in Spring Boot:*

* *Purpose: Filters in Spring Boot are used to perform tasks such as logging, authentication, authorization, request/response modification, and more.*
* *Implementation: Filters can be implemented by implementing the javax.servlet.Filter interface or extending the javax.servlet.GenericFilterBean class.*
* *Filter Configuration: Filters can be configured in different ways, such as using annotations (@WebFilter), FilterRegistrationBean in a configuration class, or configuring in application.properties or application.yml using spring.servlet.filter.registration properties.*
* *Order: Filters can have an order defined using the setOrder() method or the order property. The order determines the sequence in which filters are applied when multiple filters are present.*
* *URL Patterns: Filters can be mapped to specific URL patterns using addUrlPatterns() or url-patterns property. The filter will only be applied to requests matching the specified patterns.*
* *Pre-processing and Post-processing: Filters provide a doFilter() method where developers can implement pre-processing logic before passing the request to the next filter or servlet, and post-processing logic after the request has been processed.*
* *Use Cases: Filters are commonly used for tasks such as request logging, authentication and authorization, request/response modification (e.g., adding headers, modifying content), and handling cross-cutting concerns.*
* *Flexibility: Filters offer a flexible mechanism to intercept and manipulate requests and responses. They can be combined with other Spring Boot features, such as security filters, to implement complex application behavior.*
* *Customization: Developers can create custom filters by implementing the Filter interface or extending the GenericFilterBean class. This allows for implementing specific business logic or integrating with external systems.*

## *What are the major differences between RequestMapping and GetMapping?*

*RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation. Whereas GetMapping is only an extension of RequestMapping, which helps you to improve clarity on requests.*

## *What do you mean by hot-swapping in Spring Boot?*

*It is a way to reload the changes without restarting Tomcat, or Jetty server. Eclipse and Many other IDEs support bytecode hot swapping. If you make any changes that don’t affect the method signature, it should reload without side effect.*

## *What is @crossOrigin*

*@CrossOrigin is an annotation in Spring Boot that enables Cross-Origin Resource Sharing (CORS) for specific controller methods or the entire controller class. It allows web browsers to make cross-origin requests, meaning requests from one domain to another. By using @CrossOrigin, you can specify which origins are allowed to access your controller methods, thereby controlling the CORS behavior for your Spring Boot application.*

## *Explain caching.*

*Caching is a memory are that temporary stores frequently accessed data that is otherwise expensive to get or compute.*

## *What is mean by spring batch?*

*Spring Boot Batch provides code reusability which is important when working with large numbers of records, including transaction management, logging, skipping, job processing statistics, and job restarts.*

## *Explain Apache Kafka.*

*Apache Kafka is an open-source messaging platform. LinkedIn develops it. Apache Kafka enables the user to build distributed applications and handle real-time data feeds. Kafka is suitable for both offline and online messaging*

## *Explain different types of dependency injection.*

## *There are two types of dependency injection in Spring Boot. They are as follows:*

## ***Constructor-based dependency injection:****It is a technique in which one class object supplies the dependency of another object.*

## ***Setter-based dependency injection:****It is a dependency injection in which the framework injects the primitive and string-based values using setter method.*

## *Internal working of set*

*The internal working of the Set interface in Java depends on the specific implementation being used. The Set interface is part of the Java Collections Framework and provides a collection of unique elements with no defined order.*

*Here are some common implementations of the Set interface and their internal workings:*

*HashSet: HashSet is a widely used implementation of the Set interface. It internally uses a hash table to store the elements. When an element is added to a HashSet, its hash code is computed, and the element is stored in a bucket corresponding to that hash code. If multiple elements have the same hash code, they are stored in the same bucket as a linked list. HashSet provides constant-time performance for the basic operations like adding, removing, and checking for the presence of an element.*

*TreeSet: TreeSet is an implementation of the Set interface that provides a sorted set of elements in ascending order. It internally uses a balanced binary search tree, typically a red-black tree, to store the elements. The elements are ordered based on their natural ordering or a custom comparator provided at the time of creation. TreeSet provides logarithmic-time performance for basic operations like adding, removing, and searching for an element.*

*LinkedHashSet: LinkedHashSet is an implementation of the Set interface that maintains the insertion order of elements. It internally uses a combination of a hash table and a linked list to store the elements. The hash table provides constant-time performance for adding, removing, and searching for an element, while the linked list maintains the order of insertion.*

*EnumSet: EnumSet is a specialized implementation of the Set interface specifically designed for use with enum types. It is highly optimized for enum values and provides constant-time performance for basic operations. Internally, EnumSet uses a bit vector to represent the set of enum values.*