Thursday, February 8, 2024 4:32 PM

1) Immutablity of a class(Show an example in IDE)
Immutable objects means content cannot be changed
All data fields are private
There can't be any setter methods
No getter methods can return a data field that is mutable

2) How will you sort a salary of an Employee? (Show in IDE)

```
List<Integer>arr=List.of(234488888 ,4567 ,456734567 ,4566777 );

//printsalaryinreverseorder
arr.stream().sorted(Comparator .reverseOrder()).forEach(System.out::println);

//printsalaryinascedingorder
arr.stream().sorted((a,b)->a.compareTo(b)).forEach( System.out::println);
Collections .sort(arr,Collections .reverseOrder());
```

3) Once Sort Remove the 4th Highest Salary of an Employee? (Show in IDE)
Use index position after sorting
4) What are Generics?

5) Are you aware about the Design Pattern? (if yes then explain )
6)What is Multithreading, How you used that in your project?
7) What are Concurrent HashMap?
It is a thread safety & also it manages the concurrent update from multiple threads
It uses fine-grained locking mechanisms to allow multiple threads to access and update different segments of the map concurrently.
Segmentation: Internally, ConcurrentHashMap is divided into multiple segments, each of which acts as an independent hash table. This segmentation reduces contention by allowing multiple threads to access different segments concurrently, improving overall performance in highly concurrent scenarios.
8) What are HashMap?

- HashMap is not thread safe & one null value of key and value is allowed

9) Merge Sort
DFS & BFS

- Design Singleton class ? prevent it from reflection, cloning, serialization
- Explain OOPS concepts . rules of overriding.
- Implementation internal of HashMap ? explain in detail about hashing
- Hashmap is implemented in using hashing technique , For example if the any input comes with key value pair then hashcode is calculated basis on key and then index will be determined basis on hash code . It has 4 parts one for key , one for value ,one for hashcode and other part is for reference to next node.

If any new value will end up with same index then it will stored as linked list.

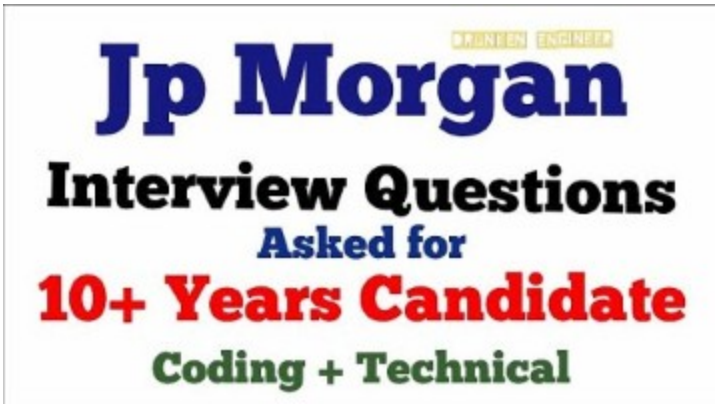
- difference b/w interface and abstract class ? when to use what ?
- Any class it contains the abstract method is called abstarct class , WE need to use abstract class if requirement is not 100 % and it may extend .
- Interface is something like 100 % inheritance where we can achieve multiple inheritance
- write sql queryto get employee name with manager

1. What are the issue might appear in multithreading?
2. Concurrent Hashmap how it works? why is better than synchronised block?
3. how to achieve locking? difference between lock and shynchronised
4. Design patterns related questions ( example of various design pattern)
5. how to secure micro services in Spring?
6. how to optimise a query if you find out its slow?
7. difference between arraylist and linkedlist, which one is better?
8. Gave me a scenario and asked me how to solve this?
9. Different Joins in SQL.
10. Future, callable, executor service.
11. S.O.L.I.D principle with examples.

Comparator and Comparable
Continuous integration
Can we call JavaStoreProcedure from Sprng boot
Convert Multiple to linear list -- We may use flat map
Volatile key words
Aggegation in Java ,
Which class contain clone method
AOP
Jconsole --

Hibernate is ORM( Object Relational Mapping ) tool used to map the Java objects to database table.
JPA is an interface in Hibernate
It supports HQL ( Hibernate Query language)
It supports implicit t transaction management
Hibernate supports caching for better performance
Importance interface in are session factory , transaction
There are 3 caches , First Level, Second level and Query Cahce
Second level cahce has EHCAche

JP Morgan Interview Questions | Interview Experience | 10 + years




What are solid design principals.

Comparable and comparator difference
Difference between stack and heap memory
Importance of equal and hand code

When array object is created the load factor will be 0.75 and size is 10
Order is maintained means the order we add the order we retrieve , When 7th item is inserted then recalculation of size will happen .The formula is below
New capacity = initial capacity * 3/2 +1;

10 * 3/2 +1 = 16 - New capacity is 16

How to generate the singleton and immutable class

LinkedList:

Use a standard LinkedList when you need a linear collection of elements.
Suitable when you want to represent a sequence of data where each element points to the next one and the last element points to null.
It's commonly used in scenarios where you frequently insert or delete elements from the list, as it provides efficient insertion and deletion operations.
CircularLinkedList:

Use a CircularLinkedList when you need a list where the last node points back to the first node, forming a loop.
Suitable for applications like round-robin scheduling, where you need to cycle through a list repeatedly.
CircularLinkedList can be useful in implementing data structures like circular buffers or queues.
It's not often used in general-purpose scenarios but can be beneficial in specific situations where circular traversal or looping is required.

Observer Design pattern

Different scope of spring

1. Singleton Scope:
   - The default scope in Spring.
   - Only one instance of the bean is created per Spring container (per ApplicationContext).
   - All requests for the bean result in the same instance being returned.
   - It is suitable for stateless beans that are safe to share across multiple threads, such as DAOs, services, etc.
2. Prototype Scope:
   - A new instance of the bean is created every time it is requested from the Spring container.
   - It is suitable for stateful beans or beans that require a new instance for each request, such as controllers, validators, etc.
   - Note: Spring does not manage the complete lifecycle of prototype beans. Clients are responsible for managing the lifecycle, including destruction.
3. Request Scope:
   - A new instance of the bean is created for each HTTP request.
   - It is only valid in the context of a web-aware Spring ApplicationContext.
   - This scope is suitable for beans that need to be scoped to a particular HTTP request, such as web controllers.
4. Session Scope:
   - A new instance of the bean is created for each HTTP session.
   - It is only valid in the context of a web-aware Spring ApplicationContext.
   - This scope is suitable for beans that need to be scoped to a particular HTTP session, such as user session data.
5. Application Scope (formerly known as "Global Session" in older versions of Spring):
   - A single instance of the bean is created per ServletContext.
   - It is only valid in the context of a web-aware Spring ApplicationContext.
   - This scope is suitable for beans that need to be scoped to the entire application, such as application-wide settings or resources.

Different ways of injection

1. Constructor Injection:
   Dependencies are injected via the constructor of the bean.
   This is the most recommended and commonly used form of injection as it ensures that all required dependencies are provided during object creation.
   Example:
   public class MyClass {

```
    private final Dependency dependency;

    public MyClass(Dependency dependency) {
        this.dependency = dependency;
    }p
}
```

2. Setter Injection:
   Dependencies are injected via setter methods of the bean.
   This approach allows for optional dependencies or circular dependencies.
   Example:

```
public class MyClass {
    private Dependency dependency;

    public void setDependency(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

3. Field Injection:
   - Dependencies are injected directly into fields of the bean using @Autowired annotation.
   - While it's concise, it's considered less preferable than constructor or setter injection as it limits testability and can hide dependencies.
   - Example:

```
public class MyClass {
    @Autowired
    private Dependency dependency;
}
```

4. Method Injection:
   - Dependencies are injected into methods of the bean.
   - This approach is less common and generally used for injecting dependencies that vary at runtime.
   - Example:

```
public class MyClass {
    private Dependency dependency;

    @Autowired
    public void setDependency(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

5. Interface Injection:
   - Less common in Spring, this approach involves implementing an interface that defines a method for injecting dependencies.
   - The container then provides an implementation of this interface to perform injection.
   - Example:

```
public interface DependencyInjector {
    void injectDependency(Dependency dependency);
}

public class MyClass implements DependencyInjector {
    private Dependency dependency;

    @Override
    public void injectDependency(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

6. Qualifier Annotation:
    - When there are multiple beans of the same type,@Qualifier annotation can be used to specify which bean should be injected.
    - It works in conjunction with other injection methods like field, constructor, or method injection.
    - Example: public class MyClass {
      @Autowired
      @Qualifier("specificDependency")
      private Dependency dependency;
      }

What are stereotype annotation

HashcodeKey & n-1
How to Dispable specific AutoCOnfiguration class
@EnableAutoconfiguration( exclude ={className})

Marker interface
Application resilience means continue to function dispite of failure
Serverless services in Azure needs to be read
Logging service in Azure to monitor App

Monday, February 12, 2024 12:35 PM

Setter Injection and constructor injection

If we do auto wire with any service class and specify the property required is false still spring boot does not starts in case of constructor injection

# TCS

Friday, March 1, 2024 7:01 PM

1. Previous experiences
2. Dot Net Design patterns / SOLID design principle
3. MVC and Microservices architecture

4. Multi Tenant architecture in Microservices

Multi-tenancy in microservices refers to the architectural approach where a single instance of an application serves multiple tenants (clients or customers), each with their own isolated data, configuration, and user interface. This architecture allows for efficient resource utilization and scalability while maintaining logical separation between tenants. Here's how you can implement multi-tenancy in a microservices architecture:

5. Azure Services
6. Messaging queue
7. Active directory and JWT Token based authentication
8. REST APIs
9. Design the architecture of a online survey application using microservices
10. Docker Containers
11. Circuit Breakers
12. How to handle faults and exceptions in microservice architecture
13. Performance improvement strategies in microservice.
14. Scaling and load balancing