Java

Design patterns:

1.Creational Design Pattern

Factory Pattern

1. Abstract Factory Pattern- This design patters is implemented using interface that implements various createobject method and the class implementing this interface returns diverse object depending on the keyword/condition.
2. Singleton Pattern-Default Pattern of Spring. Static method used to return only one object of the class. Can used in instances used for configuration and default variables that are required globally.
3. Prototype Pattern- This type is used when one Object is used to make objects of same type.This is achieved using deep cloning.
4. Builder Pattern- This pattern is used when we want to create a object without passing all the arguments withn a constructor. At the class level we create setters with return type this, and at the end we create a get method that return the final object with new keyword.

Structural design Pattern

Proxy Pattern

Behavorial Design Pattern

Difference between List and Array List:

List is a interface and ArrayList/LinkedList are implementation of it. We Only Create objectes of type ArrayList and LinkedList type.

We use @**Transient** keyword to protect variables to prevent them from being written or cahched. Implements Serializable to get it written in Byte form.

We use @Transient key word in java to remove passwords and sensitive data from getting written into cache or Disk

Working of Hashmap in java

1.Hashcode calculation of key using hashcode method, we can also override the hashcode method.

2.creation of an index based array whose index is fetched by the hashcode method. Array is initialzed with some Initial capacity =number of buckets default is 16. Load factor is a measure when to increase the capacity of the hashmap by 2.

3.New node is created that contains key and value. Incase the hashcode’s node contains same key, the object is replaced by new one. In case the Object keys are different the new node is added as the part of linked list.

4. POST JDK 8, the linked list has been changed to Self balancing tree.

Dynamic Method Dispatch- Overriding a class method and changing methods call by referring different objects of the same class.

Copy Constructor is basically assigning values of to the variable class from object of other class

Constructor Chaining- Calling constructor of own class by parameter using this keyword

Marker Interface-Blank interface E.G Serializable.

**Limitations of Interface:**

-Cant have Constructor, Static Variable,method implementation Members are public by default.

#Enums can have primitive data types and constructors as well

Read about Optional Class

How to get date

Fatures of java 8

Remove duplicates

Group by

Java 8 NEW Features :-

1. forEach() method in Iterable interface
2. Optional Class
3. default and static methods in Interfaces
4. Functional Interfaces and Lambda Expressions
5. Java Stream API for Bulk Data Operations on Collections
6. Java Time API
7. Collection API improvements
8. Concurrency API improvements
9. Java IO improvements

List<T> list=Arrays.asList

The finalize Method -

Sc.finalize() Its a method called automatically by GC. If an Exception Happens in finalize method, then the dump of the object is revived from the Memory by GC

Finally can be overrriden in Finalize method to customize cleanup like closing open connections.

OPTIONAL CLASS

The Primary reason to use Optional Class is to avoid null pointer Exception.

OPtional<T> =Object.

FAIL SAFE AND FAIL FAST

Failfast- Process of throwing *ConcurrentModificationException when there is modification in the data of the collection while iteration through the Collection. A modcount flag helps to keep a track of whether the list was modified or not.*

*Ways to make a class Immutable*

*-XML- change mutable to false*

*-use @immutable class as annotaion*

*-Use private variable and no setters.*

*FailSafe is the process where the Iterator does not throw any exception because the iteration happens on the copy of the collection.*

*Both Nested classes and Interface are possible in java*

*Varargs in Java*

*public void myMethod(String... strings){*  
 *// method body*  
*}*

UNDERSTAND JAVA.LANG

**SOLID PRINCIPLES – 5 principles**

**S-**single responsibility – A class should have only one reason to change.

E.g a class is having multiple functionalities and later we want to modify the methods in it, there must not be more than one method that needs to be changed.

O- Open for Extension but closed for modification- DO NOT MODIFY A TESTED CLASS instead extend it and do modifications in the other class. Method 2- add a additional method in the interface and implement in the other class.

L- Liskov substitute Principle- It adds the principle of substitution of one class with the other without affecting the function call of the calling class.

E.g a class takes another class’ as parameter and calls a method within that, in case the method gets changed the internal method must not throw any exception.

I-Interface Segmented Principle – The Interface must be divided into such a small parts that any class must not be forced to implement the methods that are not required by it.

D-Dependency Inversion Principle-

This implements the fact that the objects created within a class must capture the Interface objects rather than concrete class objects. This feature provides the flexibility to pass any of the classes that are implementing the declared Interface.

Hibernate Types of Transaction :-

-begin commit rollback is alive etc

Get –Instant Hit, Eager loading- Null

Load-hit when getter called,Lazy- Object not found exception

Save –Returns Primary Key, can save even after transaction commit(outside boundary).Only by hibernate

Persist-Returns Void, Limited boundary. Also by JPA Repo

Steps to use Entity Manager in JPA-

@PersistenceContext

**private** EntityManager entityManager;

|  |  |
| --- | --- |
| **Coupling** | **Cohesion** |
| Coupling is also called Inter-Module Binding. | Cohesion is also called Intra-Module Binding. |
| Coupling shows the relationships between modules. | Cohesion shows the relationship within the module. |
| Coupling shows the relative **independence** between the modules. | Cohesion shows the module's relative **functional** strength. |
| While creating, you should aim for low coupling, i.e., dependency among modules should be less. | While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system. |
| In coupling, modules are linked to the other modules. | In cohesion, the module focuses on a single thing. |

Next Top