Java Interview Question

Multiple DB connection:

<https://howtodoinjava.com/spring-boot2/datasource-configuration/>

Deep cloning ways ?

<https://www.studytonight.com/java-examples/how-to-make-a-deep-copy-of-an-object-in-java>

Example of comparator and comparable?

<https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/>

**1)why java use primitive?**

As we’ve seen, the primitive types are much faster and require much less memory. Therefore, we might want to prefer using them. On the other hand, current Java language specification doesn’t allow usage of primitive types in the parametrized types (generics),  in the Java collections or the Reflection API.

So whenever its not necessary we should go for primitive data types.

**2)how serialization works?**

Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

Only the objects of those classes can be serialized which are implementing **java.io.Serializable** interface. Serializable is a **marker interface** (has no data member and method).



**Advantages of Serialization**

1. To save/persist state of an object.
2. To travel an object across a network.

**Points to remember**

1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.  
2. Only non-static data members are saved via Serialization process.   
3. Static data members and transient data members are not saved via Serialization process. So, if you don’t want to save value of a non-static data member then make it transient.   
**4. Constructor of object is never called when an object is deserialized.**

Example:

//Java code for serialization and deserialization

//of a Java object

**import** java.io.\*;

**class** Emp **implements** Serializable {

**private** **static** **final** **long** ***serialversionUID*** = 129348938L;

**transient** **int** a;

**static** **int** *b*;

String name;

**int** age;

// Default constructor

**public** Emp(String name, **int** age, **int** a, **int** b)

{

**this**.name = name;

**this**.age = age;

**this**.a = a;

**this**.*b* = b;

}

}

**public** **class** SerialExample {

**public** **static** **void** printdata(Emp object1)

{

System.***out***.println("name = " + object1.name);

System.***out***.println("age = " + object1.age);

System.***out***.println("a = " + object1.a);

System.***out***.println("b = " + object1.*b*);

}

**public** **static** **void** main(String[] args)

{

Emp object = **new** Emp("ab", 20, 2, 1000);

String filename = "shubham.txt";

// Serialization

**try** {

// Saving of object in a file

FileOutputStream file = **new** FileOutputStream(filename);

ObjectOutputStream out = **new** ObjectOutputStream(file);

// Method for serialization of object

out.writeObject(object);

out.close();

file.close();

System.***out***.println("Object has been serialized\n"+ "Data before Deserialization.");

*printdata*(object);

// value of static variable changed

object.*b* = 2000;

}

**catch** (IOException ex) {

System.***out***.println("IOException is caught");

}

object = **null**;

// Deserialization

**try** {

// Reading the object from a file

FileInputStream file = **new** FileInputStream(filename);

ObjectInputStream in = **new** ObjectInputStream(file);

// Method for deserialization of object

object = (Emp)in.readObject();

in.close();

file.close();

System.***out***.println("Object has been deserialized\n"+ "Data after Deserialization.");

*printdata*(object);

// System.out.println("z = " + object1.z);

}

**catch** (IOException ex) {

System.***out***.println("IOException is caught");

}

**catch** (ClassNotFoundException ex) {

System.***out***.println("ClassNotFoundException" +" is caught");

}

}

}

**3)what is the use of serial versionUID?**

**We use the serialVersionUID attribute to remember versions of a Serializable class to verify that a loaded class and the serialized object are compatible. While desterilizing its checks the serial number which is same or not.**

**private** **static** **final** **long** serialVersionUID = 1234567L;

**what is mean by overloading and overriding**

**4)how break singleton class**

->Cloning(implement cloneable interface and override clone method and return CloneNotSupportedException),

->Deserialization(implement protected Object readResolve() method and return object in this method),

->reflection(add null check constrain in constructor and

**private** Singleton() { **if** (*instance* != **null**) { **throw** **new** RuntimeException("You hv broken Singleton class!");} //to prevent from reflection })

**5)what is JIT compiler(Just in time compiler)**

 improving the performance of Java programs by compiling bytecode into native machine code at run time. AOT(Ahead of time) compiler compiles the code into a native machine language

**6)how many ways to create immutable class**

* <https://www.quora.com/What-are-various-ways-to-create-an-immutable-class-in-Java>
* <https://salithachathuranga94.medium.com/implement-immutable-classes-with-java-df5b5b66ffd9>

**Writing your own immutable class:**

* The class must be declared as final -  
  So that no class can extend it & no child classes can be created.
* Declare all instance variables as private & final -  
  So that scope is within class & no one can reassign it.
* Say no to setter methods.
* Make all mutable fields as final.
* Initialise all the variables in constructor -  
  Since we don’t have any setter methods.
* Perform cloning of mutable objects while returning from getter method rather than returning actual object reference.
* **In case of Lists:**(say ArrayList) **return Collections.unModifiableList(contact);  
  or  
  return new ArrayList<>(contact);**
* Suppose there is an “Address” class and we have to make it immutable in “Employee” class, do this in Employee class:  
    
  - Declare it as:  
  **private final Address address;**- Declare getter as:  
  **public Address getAddress{  
  return new Address(address.getCity(); address.getZip());  
  }**

**Example Code:**(from GFG)

1. final class Employee {  
    private final String empName;  
    private final int age;  
    private final Address address;  
    private final List<String> phoneNumbers;  
    private final Map<String, String> metadata;  
     
    public Employee(String name, int age, Address address, List<String> phoneNumbers, Map<String, String> metadata) {  
    super();  
    this.empName = name;  
    this.age = age;  
    this.address = address;  
    this.phoneNumbers = phoneNumbers;  
    this.metadata = metadata;  
    }  
     
    public String getEmpName() {  
    return empName;  
    }  
     
    public int getAge() {  
    return age;  
    }  
     
    // clone the address object public Address getAddress() throws CloneNotSupportedException {  
    return (Address) address.clone();  
    }  
     
    // deep copy the list of phone numbers public List<String> getPhoneNumbers() {  
    return new ArrayList<>(phoneNumbers);  
    }  
     
    // deep copy the map of metadata public Map<String, String> getMetadata() {  
    return new HashMap<>(metadata);  
    }  
   }  
     
   final class Address implements Cloneable {  
     
    private String street;  
    private String city;  
     
    public Address(String street, String city) {  
    this.street = street;  
    this.city = city;  
    }  
     
    public String getStreet() {  
    return street;  
    }  
     
    public void setStreet(String street) {  
    this.street = street;  
    }  
     
    public String getCity() {  
    return city;  
    }  
     
    public void setCity(String city) {  
    this.city = city;  
    }  
     
    public Object clone() throws CloneNotSupportedException {  
    return super.clone();  
    }  
     
    @Override  
    public String toString() {  
    return "{Street: " + street + ", City: " + city + "}";  
    }  
     
   }

import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class ImmutableClassDemo {  
  
public static void main(String[] args) throws CloneNotSupportedException {  
  
Address address1 = new Address("s1", "c1");  
List<String> phoneNumbers = new ArrayList<>();  
phoneNumbers.add("123345");  
phoneNumbers.add("456789");  
Map<String, String> metadata = new HashMap<>();  
metadata.put("hobby", "Watching Movies");  
Employee e = new Employee("John", 23, address1, phoneNumbers, metadata);  
  
// modificationse.getAddress().setCity("c3");  
e.getAddress().setStreet("s3");  
e.getPhoneNumbers().add("1234");  
e.getMetadata().put("skill", "Java");  
e.getMetadata().put("designation", "HR");  
  
System.out.println(e.getEmpName());  
System.out.println(e.getAge());  
System.out.println(e.getAddress());  
System.out.println(e.getPhoneNumbers());  
System.out.println(e.getMetadata());  
  
}  
}

**7)why we use interface if already have abstract class?**

We can extends only one abstract class and implements many interfaces, If you are creating functionality that will be useful across a wide range of objects go for interface.

**8)can we write lambda expression for override method inside the class which is implemented into the interface**

->

interface MyInterface {

    void myMethod();

}

class MyClass implements MyInterface { //**java 7 way override to implement**

    @Override

    public void myMethod() {

        // Your implementation here

        System.out.println("Override method using lambda expression");

    }

}

public class Main {

    public static void main(String[] args) {

        MyInterface myObject = new MyClass();

**// Using a lambda expression to override the method**

        MyInterface myLambdaObject = () -> System.out.println("Override method using lambda expression");

        myObject.myMethod(); // Calls the overridden method in MyClass

        myLambdaObject.myMethod(); // Calls the overridden method using a lambda expression

    }

}

**9)can we override static method**

->Yes we can override but then its not overriding actually its called method hiding concept

**10)when unreachable exception arrives in our application**

* Have a return statement before them
* Have an infinite loop before them
* Any statements after throwing exception in a try block

**class** GFG { **while**(**true**){

**public** **static** **void** main(String args[]) sop(“**do** someting”) }

{ Sop(“Un-reachable code”)

**int** a = 2;

**for** (;;) {

**if** (a == 2)

{

**break**;

// it will never execute, so same error will be there.

System.out.println("I want to get printed");

}

}

}

}

**11)what is multicatch block**  
A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember

->At a time only one exception occurs and at a time only one catch block is executed.

->All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.**(Child ex should come first)**

**12)what is marker interface how we use in our application**

Interface which has no data member and method.  It is used to deliver information at runtime to the JVM so that it can take some action based on the information received. It tags specific meaning to class.

**13)what is mean by try with resources?**  
In Java, the try-with-resources statement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

Example:

// Writing data to a file using FileOutputStream by passing input file as a parameter

**try** (FileOutputStream fos = **new** FileOutputStream("outputfile.txt");

// Adding 2nd resource

// Reading the stream of character from

BufferedReader br = **new** BufferedReader(**new** FileReader("gfgtextfile.txt"))) { }

**catch**{}

**finally** {}

**14)can we create our own resource and close using try with resource**

->Yes we can create our own try with resource by implements AutoCloseable interface

Example:

**public** **class** **MyResource** **implements** **AutoCloseable**

{

@Override

**public** **void** **close**() **throws** Exception {

System.out.println("Closed MyResource"); } }

[https://www.baeldung.com/java-try-with-resources#:~:text=Support%20for%20try%2Dwith%2Dresources,to%20implement%20the%20AutoCloseable%20interface.](https://www.baeldung.com/java-try-with-resources%23:~:text=Support%20for%20try-with-resources,to%20implement%20the%20AutoCloseable%20interface.)

**15)internal working of Arraylist in java 8**

Internal Working of array list in java7 : <https://www.youtube.com/watch?v=SHwVUEYcXUA>

**16)internal working of hashset :**

**17)internal working of HashMap**

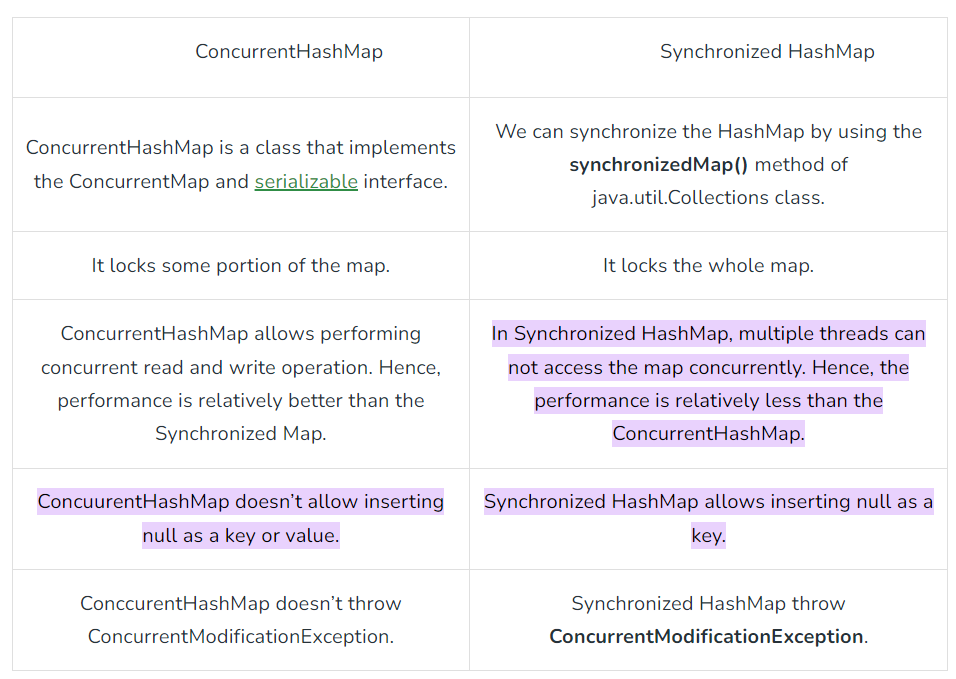
Internal working:

[https://www.youtube.com/watch?v=7k0VYHuUF6g&t=646s](%20https:/www.youtube.com/watch?v=7k0VYHuUF6g&t=646s)

**Internal working of put method in hashmap**

<https://www.youtube.com/watch?v=fSjxhOYPBRI>

**18)Difference between Synchronized Map and Concurrenthashmap**



**19)can we override equals method for content comparison in normal java class**

yes we can override ,but just equals just overriding equals method is not enough to precise content comparison, its may return wrong output, hence we have to override equals and hash-code methods for actual content comparison

**20)Explain contract between hashCode and equals method**

The basic rule of the contract states that if two objects are **equal** to each other **based** on **equals()** method, **then** the **hash code** must be the **same**, **but** if the **hash** code is the **same**, **then** **equals**() can **return false**.

**21)How many pillars of oops explain each pillar in detail**

4 pillars(Abstraction, Inheritance, Encapsulation, Polymorphism)

**22)how many methods available in object class**

tostring() method

hashCode() method

equals(Object obj) method

finalize() method

getClass() method

clone() method

wait(), notify() notifyAll() methods

**23)what is immutable**

An object is immutable when its state doesn't change after it has been initialized. For example, String is an immutable class and, once instantiated, the value of a String object never changes.

**24)why string is immutable**

Apart from security aspects , String is immutable as per below aspects

1)SCP

2)Multithreading -> No external synchronized thread is required Different threads can access a single "String instance"

**3)collections: In case of hashTable & Hashmap if Keys are String and**

**25)difference between String buffer and String Builder String**

**String**

String is a predefined final class present in the java.lang package and introduced from JDK version 1.0.

String is immutable in nature.

String is Thread safe.

String class has overridden three method from object class 1)toString(), 2)hashcode(), 3)equals()

String objects can be created using new and without new keyword (literal)

String (+) operator can be used for the concatenation.

**StringBuffer:**

StringBuffer is **mutable** in nature, **Thread safe**.

In StringBuffer only the toString() method is overridden from the Object class.

StringBuffer objects can be created using the new keyword,we can’t use the literal method.

In StringBuffer we can’t use the (+) for the concatenation ,instead of that we should use the concat() method.

**String Buider:**

StringBuilder is a predefined final class present in the java.lang package and introduced from JDK version 1.5.

StringBuilder is mutable in nature.

StringBuilder is **not Thread safe.**

In StringBuilder only the toString() method is overridden from the Object class.

As same as StringBuffer ,StringBuilder objects can be created using the new keyword,we can’t use the literal method (without the new keyword).

IN StringBuilder we can’t use the (+) for the concatenation ,instead of that we should use the concat() method.

**26)why Overriding is called Runtime Polymorphism**

**27)why Overloading is called compile time polymorphism**

**28)what is Autoboxing and Auto unboxing**

autoboxing automatically occurs whenever a primitive type must be converted into an object, and auto-unboxing takes place whenever an object must be converted into a primitive type.

(1) Parsexxx (parseInt,parseLong)🡪 String to primitive

2) Valueof() 🡪 primitive+ string to wrapper

3) xxxvalue(intvalue) 🡪 wrapper obj to primitive

4) toString 🡪 wrapper obj + primitive to String)

Example:

Integer i = **new** ~~Integer~~(10);

// Unboxing the Object

**int** i1 = i;

// Autoboxing of character

Character gfg = 'a';

// Auto-unboxing of Character

**char** ch = gfg;

**29)what is Auto promotion**

Promoting int to long or double or float, this happens in method overloadoing

Int i = 10;

Long x = I;// auto promotion

**30)What is class loader**

1)extension loader 2)application 3)bootstrap

31)what is constructor

32)what is mean by interface

33)Difference between Interface and Abstract class

34)how to call abstract class constructor

35)how to create our own Immutable class

36)what is mean by rethrowing an exception

37)difference between throw an throws

38)What is difference between Checked and Unchecked Exception in Java

39)What are important methods of Java Exception Class

40)How to write custom exception in Java

41)What is OutOfMemoryError in Java

42)What is difference between final, finally and finalize in Java

43)In which case finally block is not executed

🡪 When the System.exit() method is called in the try block

when the program is caught in an infinite loop or other non-terminating code.

**44)What happens when exception is thrown by main method**

45)**what are the nine key interface present in collection**

🡪Collection. List. Set. SortedSet. NavigableSet. Queue. Map. SortedMap.

**46)How many methods present in collections calss (13 methods)**

sort(List list)

sort(List list, Comparator<? super Project> c)

shuffle(List<?> list)

reverse(List<?> list)

rotate(List<?> list, int distance)

swap(List<?> list, int i, int j)

replaceAll(List list, String oldVal, String newVal)

copy(List<? super String> dest, List<? extends String> src)

Collections.binarySearch(list, "element 4")

frequency(Collection<?> c, Object o)

disjoint(Collection> c1, Collection> c2)

min(Collection extends ?> coll)

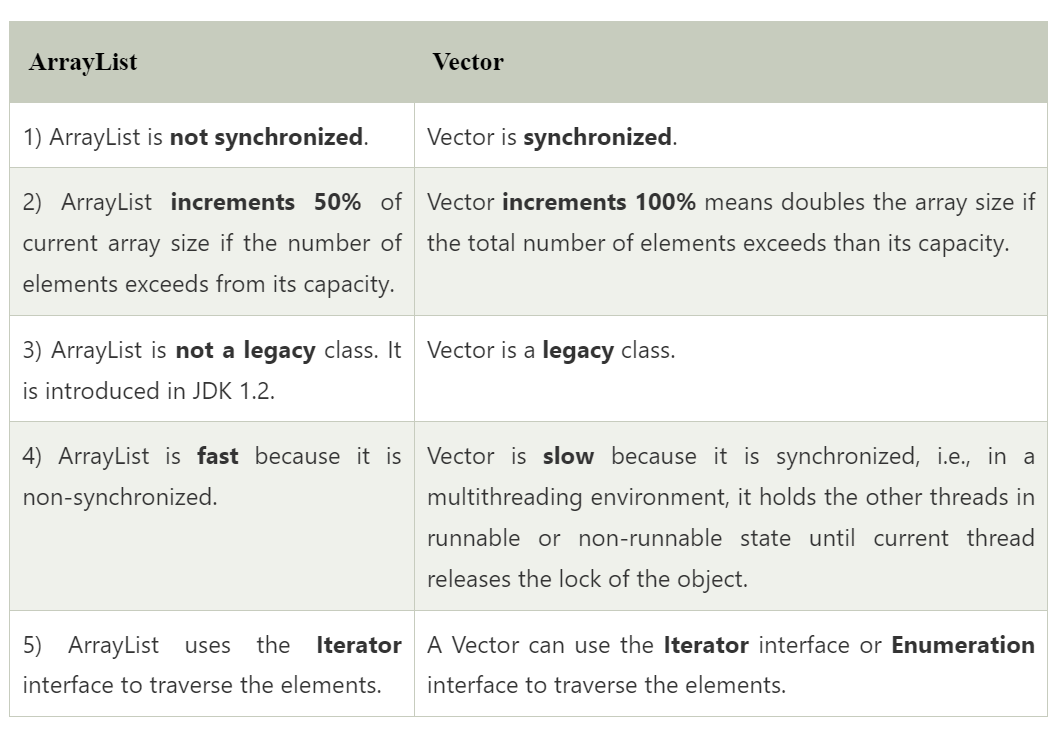
max(Collection extends ?> coll)

**47)how many types of cursor available in collection**

🡪Enumeration. iterator, listIterator

|  |  |  |  |
| --- | --- | --- | --- |
| Propery | Enumeration | iterator | listIterator |
| Legacy | Yes | No | no |
| It is applicable to | Applicable only for legacy classe(vector,stack,hashTable etc) | Only for collection object | Only for list object |
| movement | Forward direction | Forward direction | Forward & backward direction |
| How to get it | Using elements() method | Iterator() method | listIterator() method |
| accessebility | Only read | Read and remove | Read/remove/replace/add |
| method | hasMoreElements  nextElements | hasNext()  next()  remove() | hasNext(), next(), nextIndex()  hasPrevious(), previous(),  previousIndex(), add(), remove(),set() |
|  | **import** java.util.Vector;  **import** java.util.Enumeration;    Enumeration months;  Vector<String> monthNames = **new** Vector<>();    monthNames.add("November");  monthNames.add("December");  months = monthNames.elements();  **while** (months.hasMoreElements()) {  System.***out***.println(months.nextElement());} | ArrayList<Integer> al = **new** ArrayList<Integer>(); // At the beginning itr(cursor) will point to index just before the first element in al  Iterator<Integer> itr = al.iterator();  // Checking the next element where condition holds true till there is single element  // in the List using hasnext() method  **while** (itr.hasNext()) {  // Moving cursor to next element  **int** i = itr.next();  // Getting elements one by one  System.***out***.print(i + " ");  // Removing odd elements  **if** (i % 2 != 0)  itr.remove();  } | ArrayList al = **new** ArrayList();  // At beginning ltr(cursor) will point to index just before the first element in al  ListIterator ltr = al.listIterator();  // Checking the next element availability  **while** (ltr.hasNext()) {  // Moving cursor to next element  **int** i = (Integer)ltr.next();  // Getting even elements one by one  System.***out***.print(i + " ");  // Changing even numbers to odd and adding modified number again in iterator  **if** (i % 2 == 0) {  // Change to odd  i++;  // Set method to change value  ltr.set(i);  // To add  ltr.add(i);  }  } |

**48)difference Between ArrayList and vector**



49)**why ArrayList is allowed duplicate**

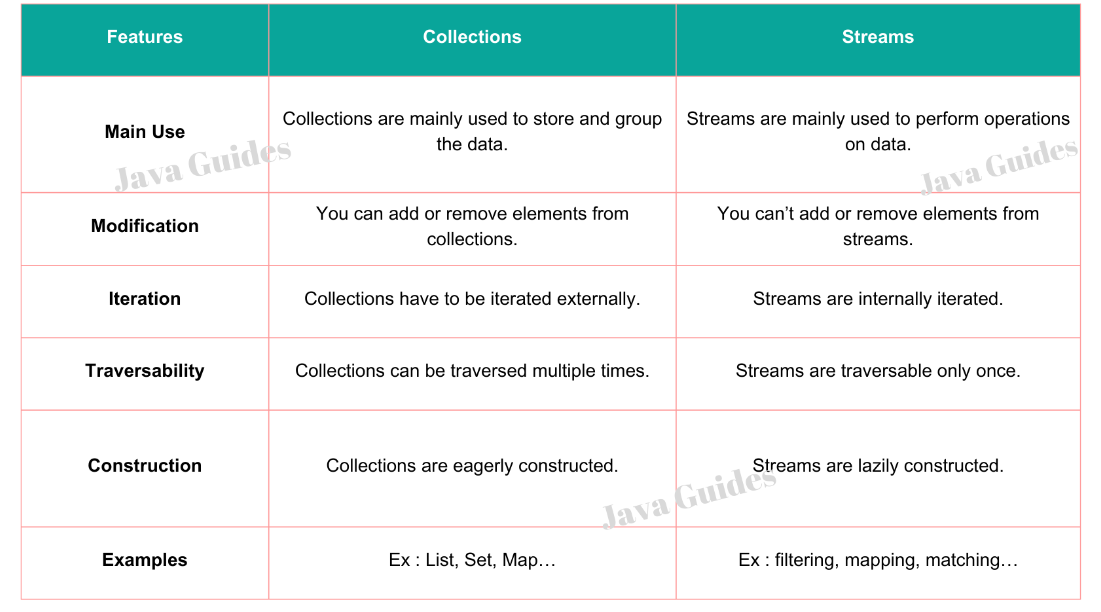
It does not use hashCode and equals method

**50)why Set is not allowed duplicate**

Internally calls map and map uses hashCode n equals

51)**what is default size of ArrayList 🡪 10**

**52)Difference between strem and collection:**



**53)What is Optional class in java?**

[**https://www.geeksforgeeks.org/java-8-optional-class/**](https://www.geeksforgeeks.org/java-8-optional-class/)

The Optional class in Java 8 is a container object which is used to contain a value that might or might not be present. It was introduced as a way to help reduce the number of [NullPointerExceptions](https://www.java67.com/2021/05/how-to-solve-nullpointerexception-in-java.html)that occur in Java code. It is a part of the java.util package and was added to Java as part of Java 8.

The Optional class provides several methods that can be used to interact with the value that is stored inside the Optional. For example, the isPresent() method can be used to check if a value is present or not. If a value is present, you can use the get() method to retrieve it.

**54)difference between map() and flatmap()**

The map() and flatmap() are two important operations in the new functional Java 8. Both represent functional operation and they are also methods in java.util.stream.Stream class but the map is used for transformation and flatmap is used for both transformation and flattening, that's why it's called the flatmap. The key *difference between map() and flatmap() function* is that when you use a [map()](http://java67.blogspot.com/2015/01/java-8-map-function-examples.html), it applies a function on each element of the stream and stores the value returned by the function into a new Stream. This way one stream is transformed into another like a Stream of String is transformed into a Stream of Integer where each element is the length of the corresponding Stream.  
  
The key thing to remember is that the function used for transformation in the map() **returns a single value**. If map() uses a function, which, instead of returning a single value returns a Stream of values then you have a Stream of Stream of values, and [flatmap()](http://java67.blogspot.com/2016/03/how-to-use-flatmap-in-java-8-stream.html)is used to flat that into a Stream of values.

**The map() Function**

The [map()](https://www.java67.com/2015/01/java-8-map-function-examples.html) function is a fundamental operation in the Streams API. It takes a Function<T, R> as an argument and applies it to each element of the stream, producing a new stream of type Stream<R>. The Function<T, R> interface has a single method, apply(T t), that takes an argument of type T and returns a value of type R. Here is an example of using the [map()](https://www.java67.com/2015/01/java-8-map-function-examples.html) function to transform a list of strings into a list of their lengths:

List<String> fruits = Arrays.asList("apple", "banana", "cherry");  
List<Integer> lengths = fruits.stream()  
 .map(String::length)  
 .collect(Collectors.toList());  
  
// lengths : [5, 6, 6]

In this example, [we create a stream from a list of strings](https://www.java67.com/2019/05/how-to-convert-stream-to-list-set-map-in-java.html)and use the map() function to transform each string into its length. The resulting stream contains the lengths of each string, which we collect into a list.

**The flatMap() Function**

The [flatMap()](https://www.java67.com/2016/03/how-to-use-flatmap-in-java-8-stream.html) function is similar to map(), but it has a different signature: it takes a Function<T, Stream<R>> as an argument and produces a stream of type Stream<R>. The Function<T, Stream<R>> interface has a single method, apply(T t), that takes an argument of type T and returns a stream of type Stream<R>. Here is an example of using the [flatMap](https://www.java67.com/2016/03/how-to-use-flatmap-in-java-8-stream.html)() function to transform a list of strings into a stream of their characters:

List<String> fruits = Arrays.asList("apple", "banana", "cherry");  
List<Character> chars = fruits.stream()  
 .flatMap(s -> s.chars().mapToObj(c -> (char)c))  
 .collect(Collectors.toList());  
  
// chars : [a, p, p, l, e, b, a, n, a, n, a, c, h, e, r, r, y]

**55)Can we use custom object as key in hashmap**

Yes we can but we, but we have to override hashcode and equals method , and make sure you are giving proper implementation of both methods

**56)Find length of strings in array**

List<String> fruits = Arrays.asList("apple", "banana", "cherry");  
List<Integer> lengths = fruits.stream()  
 .map(String::length)  
 .collect(Collectors.toList()); // lengths : [5, 6, 6]

**57)Convert String to int**

List listOfIntegers = Stream.of("1", "2", "3", "4")

.map(Integer::valueOf)

.collect(Collectors.toList());

**58)how many legacy classes availabel in List 🡪vector,stack**

**59)how many ways to create ArrayList as Synchronized**

There are two ways to create a Synchronized ArrayList.Collections. synchronizedList() method. 2. Using CopyOnWriteArrayList.

1)  List<String> list =  Collections.synchronizedList(new ArrayList<String>());

2) CopyOnWriteArrayList<String> threadSafeList= new CopyOnWriteArrayList<String>();

**60)Disadvantages of using CopyOnWriteArrayList instead of synchronizedList ?**

Slower: CopyOnWriteArrayList is slower than synchronizedList because it needs to create a copy of the underlying array every time an element is added or removed. This can be a significant overhead, especially if you are making a lot of modifications to the list.  
Not as efficient for iteration: SynchronizedList is more efficient for iteration than CopyOnWriteArrayList because it does not need to create a copy of the underlying array for each iterator. This can be a significant improvement if you are iterating over the list frequently.

**61)what is mean by Sorted List**

There is not sorted list class available in list, However

Sorting refers to arranging data in a specific order using certain criteria. You can sort different types of data, including numbers, strings, and objects. Java provides built-in methods for sorting, such as the Collections classes.

List<Integer> numbers = new ArrayList<Integer>();

numbers.add(3);

numbers.add(1);

numbers.add(4);

numbers.add(2);

**Collections.sort(numbers);**

System.out.println("Sorted List: " + numbers);

**62)what is underline data Structure of HashSet :** hashTable

**63)what is mean by TreeMap**

64)why tree set return 0,1,-1

-iterate tree map

-what is mean by comparator

-Difference between Comparabel and Compartor

-what is HashMap

-when we go for HashMap

-what is internal implimentation of HashMap

-which method is used to compare two object in HashMap : hashcode and equals

-What is Identity HashMap

-difference between HashMap and Hashtabel

-Difference between HashMap and Concurrent HashMap

-why Concurrent collection comes in Picture

-what is mean by HashTabel

-Internal Implimentation of HashTabel

-What is Weak HashMap

Java preparation

All basic of java

what is java, it is purely object Oriented or not,

Feature of java, why java is platform Independent

Inner Classes

**- can we access private data member of outer in inner class**

Inner classes can access the variables of the outer class, including the private instance variables. Unlike the non-static nested classes, the static nested class cannot directly access the instance variables or methods of the outer class. They can access them by referring to an object of a class

**-if the local variable declared as final then we wont get any compile time error**

Anonymous inner class

-inner class without name

-the main purpose of anonymous innerclass is instant use

-Types of Anonymous Inner classes

-Annonymous inner class that extend class

-A class which implements interface

-A class that define inside arguments

1)Normal Inner class

Inner cant have any static declaration, cant run from command prompt.

2)Method class

3)Anonymous inner class

4)static nested class

Demo d=new Demo()

{

I ceate class that extend Demo class without name

For that child class we creating object with parent reference

}

Runnabel r=new Runnable()

{

I created a class that implements runnabel interface for that implement class i creating an objet with interface reference

}

Annonymous inner class that extend class

Final

-every final variable replaced value at compiled time

-local variabel is stored in satck

-object is stored in heap area

-

Java.lang Package

-String class

-Object class(All methods present in object class)

-Inner classes(Types of ineer classes) etc

Colletion Framework

-Hirachy of Collection

-How many Interface present in collection

-Map

-Internel working of hashmap

Concurrent colletion

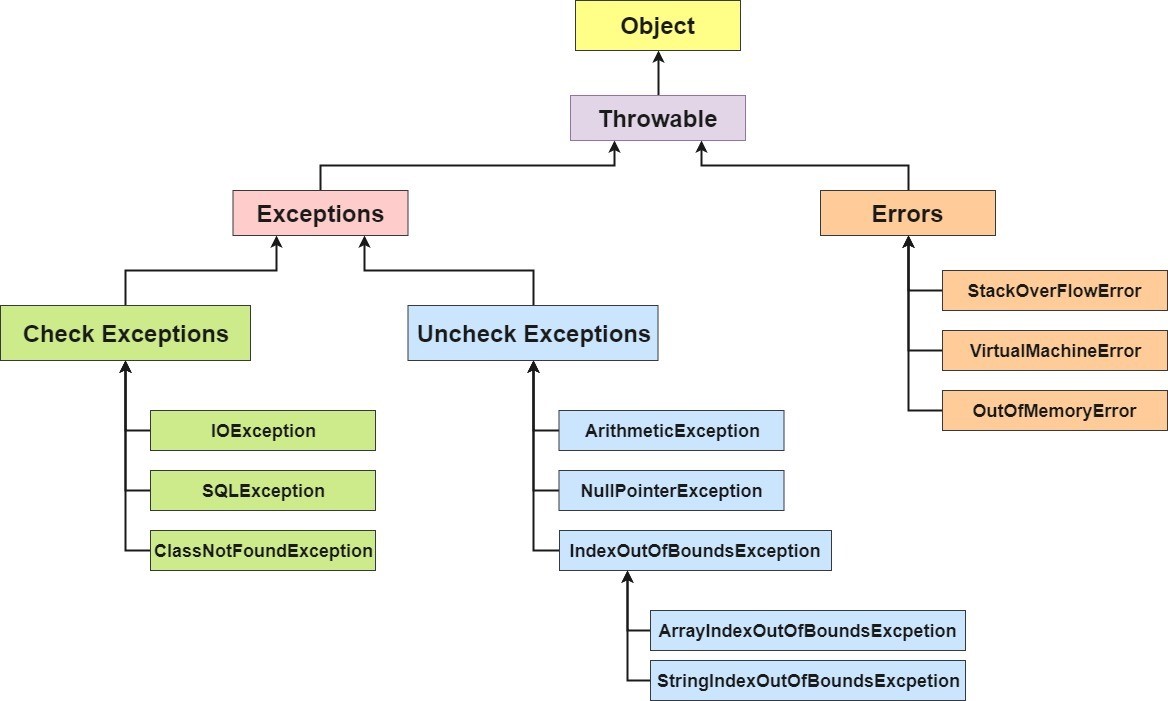
-How many ways to create Synchronized List

-difference between hashmap and Concurrent hashmap

-How to avoid ConcurrentMoificationException

-where we use concurrent collection

**EXCEPTION**



-**what is exception** : unwanted event occurs which abruptly stop/disturb flow of program, The core advantage of exception handling is **to maintain the normal flow of the application**.

-**Difference between throws and throw**

**throw** -> custom ex .. Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.

**throws** -> multiple ex at a time at method level ..Using throws keyword, we can declare both checked and unchecked exceptions. However, the

throws keyword can be used to propagate checked exceptions only.

**throwable** -> super / parent class of ex hierarchy

-**Checked and Uncheked Exception**

|  |  |
| --- | --- |
| Checked | Unchecked |
| checked at compile time | Does not checked at compile time |
| **ClassNotFoundException :** It is a kind of checked exception that occurs when a **Java virtual machine** (JVM) tries to load a class but fails because it is unable to find the classpath. Since it is a checked exception, it should be handled explicitly by using the try-catch block or by using the throws keyword.  **InterruptedException :** It usually occurs when a thread is interrupted while waiting or sleeping or occupied in some task. During this, a method interrupt() is been called by the code in the thread which blocks the I/O operations.  **IOException :** It is an input/output exception and it occurs whenever an input or output operation fails or is interpreted in java. For instance, when you try to read a file that does not exist, it will throw an IOException.  **SQLException :** This exception provides detail about SQL database error if it occurs. The error can be SQL syntax or SQL driver.  **FileNotFoundException :** It occurs when a file that we are trying to find is unavailable in the directory. It is available in class java.io and is a checked exception because it is thrown by the constructor RandomAccessFile at run time. | ***1. ArrayIndexOutOfBoundsException:****This exception is thrown when you attempt to access an array index that is out of bounds.* ***2. NullPointerException:****This exception is thrown when you attempt to access a null object reference.* ***3. ArithmeticException:****This exception is thrown when you attempt to divide by zero or perform an invalid arithmetic operation.* |
|  |  |
|  |  |

**-Difference between Exception and Error**

1)File not found ex

when we write or read file using serialisation or fileInputStream & fileOutputStream its intenally use IO exception

eg) **public** **final** **void** writeObject(Object obj)**throws** IOException

2)what type of ex is file not found- > its checked exception

3)How can u handle ex in java-> **try**, **catch** -> **throws** -> **throw**

4)when does file not found ex occurs -> an attempt to open file that is specified path name fail

5)**this** file not found ex is thrown by whom-> file input stream and file out put stream

5) why is the file not being found -> inccorect set up of file system path -> file might get corrupted -> system is trying to save the file which

does not exist

6)what is parent of file not found -> ioEx

7)how **do** i check whether the file is exist in java -> (**public** **boolean** exists())

The exists() function is a part of the File **class** in Java. This function determines whether the is a file or directory denoted by the

**abstract** filename exists or not. The function returns **true** **if** the **abstract** file path exists or **else** returns **false**.

example: File tmpDir = **new** File("/var/tmp");

**boolean** exists = tmpDir. exists();

The existing method of the Java File **class** returns **true** **if** the file or directory exists, and **false**

8)How **do** u read file in java : https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/

Java FileWriter and FileReader classes are used to write and read data from text files (they are Character Stream classes).

It is recommended not to use the FileInputStream and FileOutputStream classes **if** you have to read and write any textual information as these

are Byte stream classes.

// Creating a text File using FileWriter

**import** java.io.FileWriter;

**import** java.io.IOException;

**class** CreateFile

{

**public** **static** **void** main(String[] args) **throws** IOException

{

// Accept a string

String str = "File Handling in Java using "+

" FileWriter and FileReader";

// attach a file to FileWriter

FileWriter fw=**new** FileWriter("output.txt");

// read character wise from string and write

// into FileWriter

**for** (**int** i = 0; i < str.length(); i++)

fw.write(str.charAt(i));

System.***out***.println("Writing successful");

//close the file

fw.close();

}

}

// Reading data from a file using FileReader

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**class** ReadFile

{

**public** **static** **void** main(String[] args) **throws** IOException

{

// variable declaration

**int** ch;

// check if File exists or not

FileReader fr=**null**;

**try**

{

fr = **new** FileReader("text");

}

**catch** (FileNotFoundException fe)

{

System.***out***.println("File not found");

}

// read from FileReader till the end of file

**while** ((ch=fr.read())!=-1)

System.***out***.print((**char**)ch);

// close the file

fr.close();

}

}

To read and write JSON file in java -> use Jackson library's object mapper class

// create an instance of ObjectMapper class

# Controller

@PostMapping("/upload")

**public** ResponseEntity<String> uploadFile(@RequestPart("file") MultipartFile file,

@RequestParam String resourceSubmission) {

String[] split = resourceSubmission.split(",");

System.***out***.println(split.toString());

**if** (**null** == file.getOriginalFilename()) {

**return** **new** ResponseEntity<>(HttpStatus.BAD\_REQUEST);

}

**try** {

ObjectMapper mapper = **new** ObjectMapper();

ResourceSubmission rs = mapper.readValue(resourceSubmission, ResourceSubmission.**class**);

System.***out***.println(file.getOriginalFilename());

rs.setAttachment(file.getBytes());

rs.setSubmissonDate(LocalDate.now());

resourceSubmissionService.saveFile(rs);

} **catch** (IOException e) {

System.***out***.println(e.getMessage());

}

**return** **new** ResponseEntity<>("Good Job", HttpStatus.OK);

}

ServiceImpl

@Override

**public** ResourceSubmission saveFile(ResourceSubmission resourceSubmission) {

ResourceSubmission rs = **null**;

**try** {

rs=resourceSubmissionRepo.save(resourceSubmission);

} **catch** (Exception e) {

System.***out***.println(e);

}

**return** rs;

}

#POJO

@Entity

@Table(name = "res\_sub")

**public** **class** ResourceSubmission {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ResourceSubmissionId")

**private** **int** ResourceSubmissionId;

@Column(name = "remarks")

**private** String remarks;

@Column(name = "attachment")

**private** **byte**[] attachment;

@Column(name = "Resource\_Name")

**private** String resourceName;

@Column(name = "skills")

**private** String skills;

//@OneToMany(cascade = CascadeType.ALL,targetEntity = Employee.class)

@JoinColumn(name = "employee\_resource\_submission\_fk")

//private List<Employee> employee;

**private** **int** employeeResourceSubmissionFk;

//@OneToMany(cascade = CascadeType.ALL ,targetEntity = Requirement.class)

@JoinColumn(name = "requirement\_resource\_submission\_fk" )

//private List<Requirement> requirement;

**private** **int** requirementResourceSubmissionFk;

**private** LocalDate submissonDate;

**public** ResourceSubmission() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** ResourceSubmission(**byte**[] resume) {

**super**();

**this**.attachment = attachment;

}

**public** ResourceSubmission(**int** i, String string, String string2, String string3, String string4, **int** j) {

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@return** the employee\_resource\_submission\_fk

\*/

**public** **int** getEmployeeResourceSubmissionFk() {

**return** employeeResourceSubmissionFk;

}

/\*\*

\* **@param** employee\_resource\_submission\_fk the employee\_resource\_submission\_fk to set

\*/

**public** **void** setEmployeeResourceSubmissionFk(**int** employeeResourceSubmissionFk) {

**this**.employeeResourceSubmissionFk = employeeResourceSubmissionFk;

}

/\*\*

\* **@return** the resourceSubmissionId

\*/

**public** **int** getResourceSubmissionId() {

**return** ResourceSubmissionId;

}

/\*\*

\* **@param** resourceSubmissionId the resourceSubmissionId to set

\*/

**public** **void** setResourceSubmissionId(**int** resourceSubmissionId) {

ResourceSubmissionId = resourceSubmissionId;

}

/\*\*

\* **@return** the remarks

\*/

**public** String getRemarks() {

**return** remarks;

}

/\*\*

\* **@param** remarks the remarks to set

\*/

**public** **void** setRemarks(String remarks) {

**this**.remarks = remarks;

}

/\*\*

\* **@return** the attachment

\*/

**public** **byte**[] getAttachment() {

**return** attachment;

}

/\*\*

\* **@param** attachment the attachment to set

\*/

**public** **void** setAttachment(**byte**[] attachment) {

**this**.attachment = attachment;

}

/\*\*

\* **@return** the resourceName

\*/

**public** String getResourceName() {

**return** resourceName;

}

/\*\*

\* **@param** resourceName the resourceName to set

\*/

**public** **void** setResourceName(String resourceName) {

**this**.resourceName = resourceName;

}

/\*\*

\* **@return** the skills

\*/

**public** String getSkills() {

**return** skills;

}

/\*\*

\* **@param** skills the skills to set

\*/

**public** **void** setSkills(String skills) {

**this**.skills = skills;

}

/\*\*

\* **@return** the requirement\_resource\_submission\_fk

\*/

**public** **int** getRequirementResourceSubmissionFk() {

**return** requirementResourceSubmissionFk;

}

/\*\*

\* **@param** requirement\_resource\_submission\_fk the requirement\_resource\_submission\_fk to set

\*/

**public** **void** setRequirementResourceSubmissionFk(**int** requirementResourceSubmissionFk) {

**this**.requirementResourceSubmissionFk = requirementResourceSubmissionFk;

}

/\*\*

\* **@return** the submissonDate

\*/

**public** LocalDate getSubmissonDate() {

**return** submissonDate;

}

/\*\*

\* **@param** submissonDate the submissonDate to set

\*/

**public** **void** setSubmissonDate(LocalDate submissonDate) {

**this**.submissonDate = submissonDate;

}

@Override

**public** String toString() {

**return** "ResourceSubmission [ResourceSubmissionId=" + ResourceSubmissionId + ", remarks=" + remarks

+ ", attachment=" + Arrays.toString(attachment) + ", resourceName=" + resourceName + ", skills="

+ skills + ", employee\_resource\_submission\_fk=" + employeeResourceSubmissionFk

+ ", requirement\_resource\_submission\_fk=" + requirementResourceSubmissionFk + ", submissonDate="

+ submissonDate + "]";

}

}

Repository

@Repository

**public** **interface** ResourceSubmissionRepository **extends** JpaRepository<ResourceSubmission, Integer> {

@Query(value="select \* from res\_sub rs where rs.requirement\_resource\_submission\_fk=:requirement\_resource\_submission\_fk", nativeQuery = **true**)

**public** List<ResourceSubmission> getResourceByReqId(@Param("requirement\_resource\_submission\_fk") **int** requirement\_resource\_submission\_fk);

}

9)How to search file in java

Searching files in Java can be performed using the File **class** and FilenameFilter **interface**.

1st approch

// Java Program to Search for a File in a Directory

**import** java.io.\*;

// MyFilenameFilter class implements FilenameFilter

// interface

**class** MyFilenameFilter **implements** FilenameFilter {

String initials;

// constructor to initialize object

**public** MyFilenameFilter(String initials)

{

**this**.initials = initials;

}

// overriding the accept method of FilenameFilter

// interface

**public** **boolean** accept(File dir, String name)

{

**return** name.startsWith(initials);

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args)

{

// Create an object of the File class Replace the file path with path of the directory

File directory = **new** File("/home/user/");

// Create an object of Class MyFilenameFilter Constructor with name of file which is being searched

MyFilenameFilter filter = **new** MyFilenameFilter("file.cpp");

// store all names with same name

// with/without extension

String[] flist = directory.list(filter);

// Empty array

**if** (flist == **null**) {

System.***out***.println(

"Empty directory or directory does not exists.");

}

**else** {

// Print all files with same name in directory

// as provided in object of MyFilenameFilter

// class

**for** (**int** i = 0; i < flist.length; i++) {

System.***out***.println(flist[i]+" found");

}

}

}

}

// Java Program to Search for a File in a Directory

**import** java.io.File;

**public** **class** Main {

**public** **static** **void** main(String[] argv) **throws** Exception

{

// Create an object of the File class

// Replace the file path with path of the directory

File directory = **new** File("/home/user/");

// store all names with same name

// with/without extension

String[] flist = directory.list();

**int** flag = 0;

**if** (flist == **null**) {

System.out.println("Empty directory.");

}

**else** {

// Linear search in the array

**for** (**int** i = 0; i < flist.length; i++) {

String filename = flist[i];

**if** (filename.equalsIgnoreCase("file.cpp")) {

System.out.println(filename + " found");

flag = 1;

}

}

}

**if** (flag == 0) {

System.out.println("File Not Found");

}

}

}

10) Difference between **throw** , **throws** and throwable

**throw** -> custom ex

**throws** -> multiple ex at a time at method level

throwable -> **super** / paret **class** of ex hierarchy

11) Difference between **final**, **finally** and finalize

**final** -> access modifier (variable, method, **class**) ->

**class** cant inerit ->can create object of **final** **class**

variable-> eg **final** var = "name" -> If you make any variable as **final**, you cannot change the value of **final** variable(It will be constant).

method -> u can not overrride

**final** object -> eg) **final** Employee emp = **new** Employee() -> ref is **final** but we can change its variable values

eg)

**class** Demo

{

**int** value = 10;

**public** **void** setValue(**int** x)

{

value = x;

}

**public** **int** getValue()

{

**return** value;

}

}

**public** **class** FinalObjectExample1

{

**public** **static** **void** main(String args[]) //throws Throwable

{

**final** Demo demo = **new** Demo();

demo.setValue(11);

System.out.println("The value is: "+demo.getValue());

}

}

12)What **do** u mean by **class** cast ex

It is run time ex - when JVM is unable to cast obj of one type to another type, **we can avoid it by using generics types**

**public** **class** ClassCastExceptionExample {

**public** **static** **void** main(String[] args) {

Object obj = **new** String("Hello");

System.***out***.println((Integer) obj);

}

}

13)**try** with resource

The **try**-with-resources statement ensures that each resource is closed at the end of the statement execution. If we don’t close the resources,

it may constitute a resource leak and also the program could exhaust the resources available to it.

in such **case** get Exception of Cannot delete file from a location

**public** String uploadFile(MultipartFile file, Employee employee,String employee\_profile\_path) {

String f = **null**;

**try** (InputStream stream = file.getInputStream()){ // try-with-resources

System.out.println(employee);

**long** id = employee.getYashEmpId();

String empId = Long.toString(id);

String UPLOAD\_DIR = employee\_profile\_path+ "/" + empId + "/";

File files = **new** File(UPLOAD\_DIR);

System.out.println(file.toString());

**if** (!files.exists()) {

System.out.println("if directory not exist then create");

**if** (!files.mkdirs()) {

System.out.println("folder created");

}

} **else** {

System.out.println("failed to create folder");

}

String filePath=UPLOAD\_DIR+(employee.getEmployeeName()+"\_"+empId)+file.getOriginalFilename().substring(file.getOriginalFilename().lastIndexOf("."),file.getOriginalFilename().length());

Files.copy(stream,Paths.get(filePath));

f = filePath;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** f;

}

14)Can we have empty **catch** block

**try** {

} **catch** (Exception e) {

}

15)what is rethrowing of exception

An exception can be rethrown in a **catch** block using **throw** keyword, **if** **catch** block is unable to handle it.

This process is called as re-throwing an exception.

**public** **class** Main {

**public** **int** test(**int** n1, **int** n2) {

**try**{

**return** n1/n2;

}**catch**(ArithmeticException e){

**throw** e;

}

}

**public** **static** **void** main(String[] args) {

Main main = **new** Main();

**try**{

System.out.println(main.test(30, 0));

}**catch**(Exception e){

e.printStackTrace();

}

}

}

16)What is exception propagation

Exception propagation is another concept of Java Exception handling. It refers to the transferring of Exceptions from one method to the previous

method in order to handle the Exception efficiently.

Exception Propagation in Unchecked Exceptions

**class** Simple {

**void** m()

{

**int** data = 50 / 0; // unchecked exception occurred

// exception propagated to n()

}

**void** n()

{

m();

// exception propagated to p()

}

**void** p()

{

**try** {

n(); // exception handled

}

**catch** (Exception e) {

System.out.println("Exception handled");

}

}

**public** **static** **void** main(String args[])

{

Simple obj = **new** Simple();

obj.p();

System.out.println("Normal flow...");

}

}

Exception Propagation in Checked Exceptions

// Java program to illustrate exception propagation

// in checked exceptions and it can be propagated

// by throws keyword ONLY

**import** java.io.IOException;

**class** Simple {

// exception propagated to n()

**void** m() **throws** IOException

{

// checked exception occurred

**throw** **new** IOException("device error");

}

// exception propagated to p()

**void** n() **throws** IOException

{

m();

}

**void** p()

{

**try** {

// exception handled

n();

}

**catch** (Exception e) {

System.***out***.println("exception handled");

}

}

**public** **static** **void** main(String args[])

{

Simple obj = **new** Simple();

obj.p();

System.***out***.println("normal flow...");

}

}

17)Do you know what is custom exception

**public** **class** EmptyCellException **extends** RuntimeException{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** EmptyCellException(String message)

{

**super**(message);

}

}

use of custom ex in serviceImpl or controller

**else** {

**throw** **new** EmptyCellException("Row : " + row.getRowNum()

+ " Contains empty cell, column number : " + cell.getColumnIndex());

}

**public** **class** ResourceNotFoundException **extends** RuntimeException{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** ResourceNotFoundException(String message)

{

**super**(message);

}

}

// get single Employee Details By Id

**public** Employee getSingleEmployeeDetails(**int** employeeId) {

**return** **this**.employeeRepo.findById(employeeId).orElseThrow(

() -> **new** ResourceNotFoundException("Resource not found with this employee Id" + employeeId));

}

18) What is unreachable **catch** block error

when we are having multiple **catch** block then exception order should be from child to parent hierarchy

**try**{}

**catch**(Exception ex){}

**catch**(IOex e) // this code will give unreachable catch block error

19) Can we thow multiple exceptions in one THROW keyword

no we can not

20)what is chained exceptions in java

In Java, a chained exception is an exception that is caused by another exception.

**class** ChainedExceptionDemo {

**public** **static** **void** main(String[] args) **throws** IOException{

**try** {

**throw** **new** IOException("IOException encountered")

.initCause(**new** EOFException("root cause is EOFException"));

} **catch** (Throwable e) {

// Handle the IOException

System.***out***.println("Caught exception -> " + e);

// Handle the EOFException here

EOFException eof = (EOFException) e.getCause();

System.***out***.println("The cause is -> " + eof);

}

}

}

21) What are the methods provided by throwable **class**

getMessage()

toString()

printStackTrace()

fillInStackTrace()

getStackTrace()

getClause()

22)Best practices in exceptions handling in java

1. Use a **finally** block to clean up resources or close them.

2. Throw specific exceptions.

3. Do not **catch** the Exception **class**. Catch the specific sub classes.

4. No exception must be thrown from the **finally** block.

5. Use descriptive messages when throwing exceptions.

23)Is it possible to override a **super** **class** method that is throwing an unchecked exception along with a checked exception

in sub-**class**

No along with unchecked we can not override checked exception

parent{

m1()**throws** nullpointerEx, IoEx/ FileNotFoundException{

}

}

child extnds parent{

m1()**throws** FileNotFoundException{

}

}

24)It is always recomended to close Db resource to keep them inside a **finally** block. and why?

The **finally** block is the block that is always executed no matter whether the exceptions are raised or not or caught.

Cleanup Operations ensure that these will always be executed irrespective of whether an exception occurs or not.

25) what does mean checked and unchecked exception

checked exception are gets check by compiler at compile time only

unchecked exception are not check by compiler at compile time

26)Custom exception example

**class** InvalidAgeException **extends** Exception

{

**public** InvalidAgeException (String str) {

// calling the constructor of parent Exception

**super**(str);

}

}

// class that uses custom exception InvalidAgeException

**public** **class** TestCustomException1

{

// method to check the age

**static** **void** validate (**int** age) **throws** InvalidAgeException{

**if**(age < 18){

// throw an object of user defined exception

**throw** **new** InvalidAgeException("age is not valid to vote");

}

**else** {

System.out.println("welcome to vote");

}

}

// main method

**public** **static** **void** main(String args[])

{

**try**

{

// calling the method

validate(13);

}

**catch** (InvalidAgeException ex)

{

System.out.println("Caught the exception");

// printing the message from InvalidAgeException object

System.out.println("Exception occured: " + ex);

}

System.out.println("rest of the code...");

}

}

27)What is OutOfmemoryError

A java.lang.OutOfMemoryError is a runtime error in Java which occurs when the Java Virtual Machine (JVM) is unable to

allocate an object due to insufficient space in the Java heap. The Java Garbage Collector (GC) cannot free up the space

required **for** a **new** object, which causes a java.lang.OutOfMemoryError.

28)can we create our own resource and close using **try** with resource

->Yes we can create our own **try** with resource by **implements** AutoCloseable **interface**

Link : https://www.baeldung.com/java-try-with-resources#:~:text=Support%20for%20try%2Dwith%2Dresources,to%20implement%20the%20AutoCloseable%20interface.

**public** **class** MyResource **implements** AutoCloseable {

@Override

**public** **void** close() **throws** Exception {

System.out.println("Closed MyResource");

}

}

geeksforgeeks

Example 2:

select e\_name,e\_salary from emp\_info a

inner join emp\_salary b

on a.emp\_id=b.emp\_id

where e\_salary in (select max(e\_salary) from emp\_salary)

**-What is spring**

Its is framework of framework,  the framework's core that controls features such as Inversion of Control and dependency injection. -

**-why spring come in picture :**

IOC, Spring MVC, DI

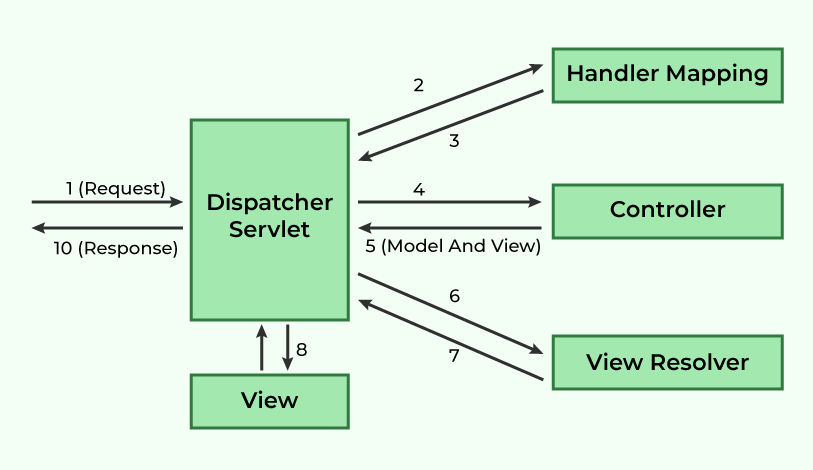
-what is dependency injection

**dependency injection** is a programming technique in which an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) or [function](https://en.wikipedia.org/wiki/Subroutine) receives other objects or functions that it requires,

types of DI: Field base , constructor base, setter base

-what is the flow of core spring

-**what is the flow of spring MVC**



-Difference between setter based injection and Constructor based Injection

-I want to print HelloWorld on console in Spring core

-**difference between @ModelAttribute and ModelandView**

- how many Stereotype Annotation present in Spring

-difference between @RestController and@controller

-what is@Component Spring

-How many Container Present in Spring

-Difference between Bean Factory and Application Context

-how to avoid Circular Dependencies Problem in Spring

-Scope Of Bean

-Have you write duplicate Id in Spring Beans.xml file

-what is View Resolver?

-what is mean by Handler Mapping is it class or interface?

-how many module present in spring?

-what is spring AOP?

-how to use Spring with JDBC

**Hibernate**

-Difference between hibernate and JDBC

-what is the advantages of hibernate

-what is drawback of JDBC

-How many core interface present in hibernate

-what is mean by Criteria

-what is mean by Query

-what is use of Criteria

-what is mean by Cache

-what is the type of cache? Difference between them

-how to enable second level cache using XML file

-what is mean by session

-what is mean by session factory

-difference between session and session factory

-what is mean by Transaction in hibernate and how to mange transaction

-difference between @Entity and @Table

-Types of Generator class

-Difference between Get an Load

-difference between Update and Save or Update

-what is mean by Projection in hibernate

-write down hibernate Configuration file

-what is cascade in hibernate

-what is types of hbm2ddl and difference between them

-how to display query on console

-what is the use of @NamedQuery

-Explain @NativeSqlQuery

-What is mean Criteria Builder

-how to generate composite primary key in hibernate

-Explain State Of Object in Hibernate

-Explain HQL in Hibernate

-difference between save and Persist

-difference between Update and Merge

-Explain Inheritance in Hibernate

-Explain relation in Hibernate @OneToOne,@OneToMany,@ManyToMany,@MayToOne

-Difference between Create and CreateAndDrop

-Difference between Delete and Truncate

Spring Boot

-what is mean by spring boot Starter

->spring boot starter is just jar file to provide auto dependency Resolution

-what is spring boot Auto Configurator

->it is used to provide Auto Configuration

-**what is spring boot Actuator**

->It is used to provide management End Point to see the application internals,Metrics, check helth of application, all end points mapping ect

-**what is mean by spring boot CLI(Command line Interface)**

->In simple words, Spring Boot CLI is Auto Dependency Resolution, Auto-Configuration, Management EndPoints,

Embedded HTTP Servers(Jetty, Tomcat etc.) and (Groovy, Auto-Imports)

Collection Interview Questions

--**how to check the element are exisist in not in ArrayList**

🡪use Arraylist.Contains(value) method

**How to add A element On Particular Index in ArrayList :** add(2, “Abc”)

-**Program to Convert HashSet to Array in Java**

🡪 String[] arr=new String[hashet.size()];

hashset.toArray(arr);

**-Java Program to Reverse ArrayList in Java**

**1st way)** // Arraylist for storing reversed elements

ArrayList<Integer> alist = **new** ArrayList<Integer>();

// Appending elements at the end of the list

alist.add(**new** Integer(12));

alist.add(**new** Integer(13));

alist.add(**new** Integer(123));

ArrayList<Integer> revArrayList = **new** ArrayList<Integer>();

**for** (**int** i = alist.size() - 1; i >= 0; i--) {

// Append the elements in reverse order

revArrayList.add(alist.get(i));

}

**2nd Way)** Collections.reverse(alist);

**3rd way)** ListIterator iterator=list.listIterator(alist.size());

**while**(iterator.hasPrevious())

{

System.***out***.println(iterator.previous());

}

-**Java Program to Reverse LinkedList in Java**

LinkedList<String> ll = **new** LinkedList<String>();

ll.add(**new** String("Physics"));

ll.add(**new** String("Maths"));

ll.add(**new** String("Java"));

ll.add(**new** String("English"));

ll.add(**new** String("Chemistry"));

**for** (**int** i = 0; i < ll.size() / 2; i++)

{

String temp = ll.get(i);

ll.set(i, ll.get(ll.size() - i - 1));

ll.set(ll.size() - i - 1, temp);

}

System.***out***.println("Reverse LL..."+ll);

**2nd Way)** Collections.reverse(ll);

-How to Iterate Tree Map

-How To Serialize ArrayList In Java With Example

[**https://howtodoinjava.com/java/collections/arraylist/serialize-deserialize-arraylist/**](https://howtodoinjava.com/java/collections/arraylist/serialize-deserialize-arraylist/)

import java.util.\*;

import java.io.\*;

public class ArrayListSerialization {

public static void main(String args[]) {

ArrayList<String> al=new ArrayList<String>();

al.add("Boston");

al.add("Dallas");

al.add("New York");

try{

FileOutputStream fos= new FileOutputStream("inputfileName");

ObjectOutputStream oos= new ObjectOutputStream(fos);

oos.writeObject(al);

oos.close();

fos.close();

}catch(IOException ioe){

ioe.printStackTrace();

}

}

}

-Deserialization of ArrayList

import java.util.\*;

import java.io.\*;

public class ArrayListSerialization {

public static void main(String args[]) {

ArrayList<String> al=new ArrayList<String>();

al.add("Boston");

al.add("Dallas");

al.add("New York");

try{

FileOutputStream fos= new FileOutputStream("inputfileName");

ObjectOutputStream oos= new ObjectOutputStream(fos);

oos.writeObject(al);

oos.close();

fos.close();

}catch(IOException ioe){

ioe.printStackTrace();

}

}

PUT and PATCH:

[https://www.linkedin.com/advice/3/what-benefits-drawbacks-using-patch-vs-put-updating#:~:text=PATCH%20is%20used%20to%20apply,if%20they%20are%20not%20modified.](https://www.linkedin.com/advice/3/what-benefits-drawbacks-using-patch-vs-put-updating%23:~:text=PATCH%20is%20used%20to%20apply,if%20they%20are%20not%20modified.)