1. Can you please tell me about yourself?
2. Can you please explain me what is your Project is all about?
3. What are your roles and responsibilities in your Project?
4. Have you worked on Production issues?
5. What will you do if team assigned Production issue to you?
6. Are you handling any teams?
7. Do you know OOPS Concepts?

Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

1. Can you please name me some of the OOPS concepts?

Objects

Classes

Object

Class

Abstraction

Inheritance

Polymorphism

Encapsulation

1. What is Encapsulation in Java?

**Encapsulation** is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

1. Why do we need public getters and setter’s methods in Java?

Getters and Setters play an important role in retrieving and updating the value of a variable outside the encapsulating class. A setter updates the value of a variable, while a getter reads the value of a variable.

First, for accessing the variables outside a class without getters/setters, we have to mark those as public, protected, or default. **Thus, we're losing control over the data and compromising the fundamental** [**OOP**](https://www.baeldung.com/cs/oop-modeling-real-world) **principle –** [**encapsulation**](https://www.baeldung.com/java-oop)**.**

Third, we cannot provide any conditional logic to the change of the variable.

Second, since anyone can change the non-private fields from outside the class directly, **we cannot achieve immutability.**

* **It helps us achieve encapsulation which is used to hide the state of a structured data object inside a class, preventing unauthorized direct access to them**
* Achieve immutability by declaring the fields as private and using only getters
* **Getters and setters also allow additional functionalities like validation, error handling that could be added more easily in the future. Thus we can add conditional logic and provide behavior according to the needs**
* We can provide different access levels to the fields; for example, the get (read-access) may be public, while the set (write-access) could be protected
* Control over setting the value of the property correctly
* With getters and setters, we achieve one more key principle of OOP, i.e., abstraction, which is hiding implementation details so that no one can use the fields directly in other classes or modules

1. Why Java is Platform independent language?

Java is platform-independent because it uses a virtual machine. The Java programming language and all APIs are compiled into bytecodes. Bytecodes are effectively platform-independent. The virtual machine takes care of the differences between the bytecodes for the different platforms. The run-time requirements for Java are therefore very small. The Java virtual machine takes care of all hardware-related issues so that no code has to be compiled for different hardware.

1. Do you know difference between JDK/JRK?

The **JRE** is the **Java Runtime Environment**. It is a package of everything necessary to run a compiled Java program, including the Java Virtual Machine (JVM), the Java Class Library, the java command, and other infrastructure. However, it cannot be used to create new programs.

The **JDK** is the **Java Development Kit**, the full-featured SDK for Java. It has everything the JRE has, but also the compiler (javac) and tools (like javadoc and jdb). It is capable of creating and compiling programs.

1. How JIT compiler works internally?

JIT compiler **only compiles the byte-code to equivalent native code at first execution**. Upon every successive execution, the JVM merely uses the already compiled native code to optimize performance.

1. Can we override finalize() method in Java? If yes, then How?

Finalize is used for garbage collection. It should not be done as we should leave the garbage collection to java itself.

1. What is Runtime Polymorphism?

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

1. Do you know Compile time Polymorphism in Java?

Whenever an object is bound with its functionality at the compile time, this is known as the compile-time polymorphism. At compile-time, java knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding.

1. Can you explain me why we need Polymorphism in Java with an example?

polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

When there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

1. What is IS-A relationship in Java and How do we achieve that?

In Java, the "IS-A" relationship refers to inheritance, where a subclass inherits properties and behaviors from its superclass. The "IS-A" relationship indicates that an object of a subclass is also an object of its superclass. For example, a Cat "IS-A" Animal, because a cat is a type of animal.

To achieve the "IS-A" relationship in Java, we use the extends keyword in the subclass declaration. Here's an example:

kotlinCopy code

public class Animal {

// properties and methods for all animals

}

public class Cat extends Animal {

// properties and methods specific to cats

}

public class Dog extends Animal {

// properties and methods specific to dogs

}

In this example, the Cat and Dog classes both inherit properties and methods from the Animal superclass. This means that we can treat a Cat object as an Animal object, because it "IS-A" type of Animal. Similarly, we can treat a Dog object as an Animal object, because it also "IS-A" type of Animal.

Using the "IS-A" relationship in Java allows us to write more generic code that can work with objects of multiple classes, as we can treat them all as objects of a common superclass.

1. Can you explain me what is inheritance in Java with an example?

See ans 18

1. Why is Java not supporting Multiple Inheritance?

Multiple inheritance occurs when a subclass inherits from more than one superclass. For example, suppose we have two classes, Class A and Class B, and we want to create a new class called Class C that inherits from both A and B. This creates a problem if both A and B have a method with the same name and signature, as Class C would not know which version of the method to use.

To avoid this issue, Java introduced the concept of interfaces, which provide a way to achieve some of the benefits of multiple inheritance without the problems. An interface is similar to a class, but it only defines method signatures without any implementation. A class can implement multiple interfaces, allowing it to inherit from multiple sources of behavior.

In addition to avoiding method name clashes, interfaces also promote a more modular and flexible programming style by allowing classes to implement only the interfaces they need, rather than being forced to inherit all the properties and behaviors of multiple superclasses.

Overall, while multiple inheritance can be useful in certain situations, it also brings with it some challenges and complexities. By using interfaces and a more modular approach to programming, Java provides a safer and more flexible alternative to multiple inheritance.

1. Top of Form
2. Bottom of Form
3. If we want to achieve Multiple Inheritance, then is there any way by which we can achieve?
4. Interfaces: As mentioned earlier, interfaces allow us to define method signatures without any implementation. A class can implement multiple interfaces, which is similar to inheriting from multiple sources of behavior. This approach allows us to achieve some of the benefits of multiple inheritance while avoiding the potential problems of naming conflicts and complexity.
5. What is the difference between Abstract and Interface?

Abstract class can have abstract and non-abstract methods. Interface can have only abstract methods. Since Java 8, it can have default and static methods also.

2) Abstract class doesn't support multiple inheritance. Interface supports multiple inheritance.

3) Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.

4) Abstract class can provide the implementation of interface. Interface can't provide the implementation of abstr

1. What is the use of Abstract Java classes?

### Abstract class in Java Important Points

1. abstract keyword is used to create an abstract class in java.
2. Abstract class in java can’t be instantiated.
3. We can use abstract keyword to create an abstract method, an abstract method doesn’t have body.
4. If a class have abstract methods, then the class should also be abstract using abstract keyword, else it will not compile.
5. It’s not necessary for an abstract class to have abstract method. We can mark a class as abstract even if it doesn’t declare any abstract methods.
6. If abstract class doesn’t have any method implementation, its better to use interface because java doesn’t support multiple class inheritance.
7. The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class.
8. All the methods in an interface are implicitly abstract unless the interface methods are static or default. Static methods and default methods in interfaces are added in Java 8, for more details read [Java 8 interface changes](https://www.digitalocean.com/community/tutorials/java-8-interface-changes-static-method-default-method).
9. Java Abstract class can implement interfaces without even providing the implementation of interface methods.
10. Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.
11. We can run abstract class in java like any other class if it has main() method.
12. What is functional interface in Java?
13. What is the difference between final, finally and finalize() in Java?

**final**: **final** is a keyword used in Java to declare a variable, method, or class that cannot be changed or overridden. When a variable is declared as **final**, it cannot be reassigned a new value after its initialization. When a method is declared as **final**, it cannot be overridden by a subclass. When a class is declared as **final**, it cannot be subclassed.

**finally**: **finally** is a block in Java that is used to execute a set of statements after the try-catch block, regardless of whether an exception is thrown or not. The **finally** block is always executed, even if there is a return statement in the try or catch block.

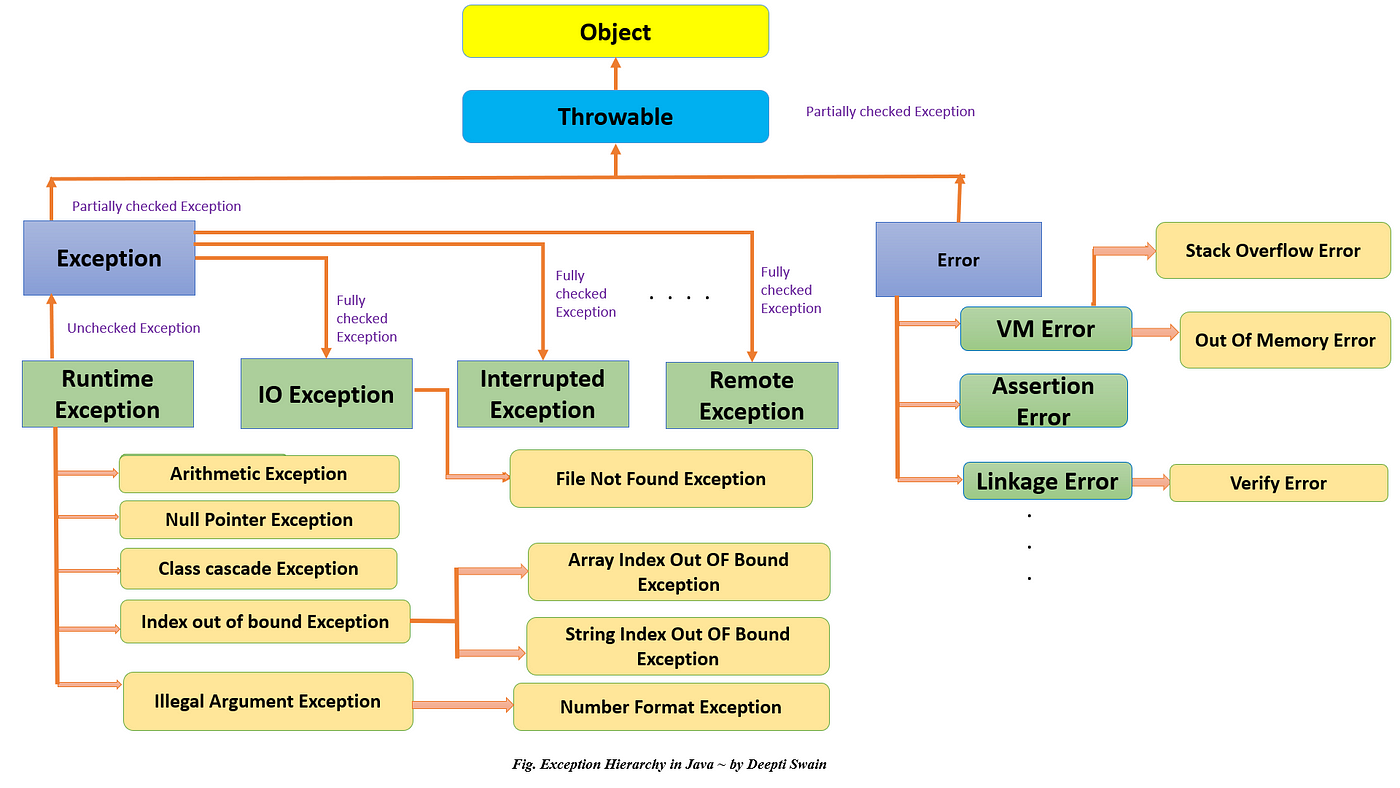
**finalize()**: **finalize()** is a method in Java that is called by the garbage collector before an object is destroyed. This method can be overridden in a class to perform some cleanup or finalization tasks before the object is destroyed. However, it is generally not recommended to rely on **finalize()** for cleanup tasks, as the timing and order of finalization is not guaranteed.

1. What are the types of Variables?

**Local variable:** This is a variable that is declared inside the body of a method.

**Instance variable:** This Java variable is defined without the STATIC keyword, but as outside of a method declaration. They are object-specific variables, which is why they are known by this name.

**Static variable:** This variable is initialized only once, just when the program execution starts. It is the variable that should be initialized first, especially before an instance variable is initialized

1. How to achieve Singleton Pattern in Java?
2. What are the types of Exceptions? 
3. Can you please explain me Checked and Unchecked Exception example in Java?

Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using [try-catch block](https://beginnersbook.com/2013/04/try-catch-in-java/) or it should declare the exception using [throws keyword](https://beginnersbook.com/2013/04/java-throws/), otherwise the program will give a compilation error.

Program:

import java.io.\*;

class Example {

public static void main(String args[])

{

FileInputStream fis = null;

/\*This constructor FileInputStream(File filename)

\* throws FileNotFoundException which is a checked

\* exception

\*/

fis = new FileInputStream("B:/myfile.txt");

int k;

/\* Method read() of FileInputStream class also throws

\* a checked exception: IOException

\*/

while(( k = fis.read() ) != -1)

{

System.out.print((char)k);

}

/\*The method close() closes the file input stream

\* It throws IOException\*/

fis.close();

}

}

Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn’t handle/declare that exception, the program won’t give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. It is up to the programmer to judge the conditions in advance, that can cause such exceptions and handle them appropriately. All Unchecked exceptions are direct sub classes of **RuntimeException** class.

class Example {

public static void main(String args[])

{

int num1=10;

int num2=0;

/\*Since I'm dividing an integer with 0

\* it should throw ArithmeticException

\*/

int res=num1/num2;

System.out.println(res);

}

}

1. Difference between Error and Exceptions in Java?

| **Errors** | **Exceptions** |
| --- | --- |
| Errors are usually raised by the environment in which the application is running. For example, an error will occur due to a lack of system resources. | Exceptions are caused by the code of the application itself. |
| It is not possible to recover from an error. | The use of **try-catch** blocks can handle exceptions and recover the application from them. |
| Errors occur at run-time and are not known by the compiler; hence, they are classified as “unchecked.” | Exceptions can be “checked” or “unchecked,” meaning they may or may not be caught by the compiler. |
| “OutOfMemory” and “StackOverflow” are examples of errors. | “IndexOutOfBounds” is an example of an unchecked exception, while “ClassNotFound” is an example of a checked exception. |

1. Try, catch and finally different patterns programs?

// Java program to demonstrate

// control flow of try-catch-finally clause

// when exception occur in try block

// and handled in catch block

class GFG

{

public static void main (String[] args)

{

// array of size 4.

int[] arr = new int[4];

try

{

int i = arr[4];

// this statement will never execute

// as exception is raised by above statement

System.out.println("Inside try block");

}

catch(ArrayIndexOutOfBoundsException ex)

{

System.out.println("Exception caught in catch block");

}

finally

{

System.out.println("finally block executed");

}

// rest program will be executed

System.out.println("Outside try-catch-finally clause");

}

}

1. Do you know try with resources in Java?

try (Scanner scanner = new Scanner(new File("testRead.txt"));

PrintWriter writer = new PrintWriter(new File("testWrite.txt"))) {

while (scanner.hasNext()) {

writer.print(scanner.nextLine());

}

}

Try with resources works as a try block has its own resources are being declare at the try block and try with resources can have multiple declaration of the resources

1. How exception chaining works in Java?

In Java, a **chained exception** is an exception that is caused by another exception. Chained exceptions are associated such that the previous exception causes each exception in the chain. It can help debug, as it can help us track down the root cause of an error.

Diagram

Description automatically generated

We can use exception chaining in situations where another exception causes one exception. However, it's important to note that chaining can make our code more difficult to read and understand. Therefore, we should use exception chaining sparingly and only when necessary.

If we use chained exceptions, it is a good idea to document the chain in our code. It'll help others understand our code and make it easier to debug if an error occurs.

For example, if an InputStream throws an IOException, and the InputStream's read() method throws an EOFException, then the EOFException is the cause of the IOException. The following code demonstrates the use of chained exceptions:

import java.io.\*;

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);

}

}

}

Output

1.78s

Caught exception -> java.io.IOException: IOException encountered The cause is -> java.io.EOFException: root cause is EOFException

1. Difference between throw and throws keyword in Java?

**Throws clause** is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.

If we see syntax wise than **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.  
For example:

throw new ArithmeticException("Arithmetic Exception");

and

throws ArithmeticException;

Throw keyword is used in the method body to throw an exception, while throws is used in method signature to declare the exceptions that can occur in the statements present in the method.

For example:  
**Throw:**

...

void myMethod() {

try {

//throwing arithmetic exception using throw

throw new ArithmeticException("Something went wrong!!");

}

catch (Exception exp) {

System.out.println("Error: "+exp.getMessage());

}

}

...

**Throws:**

...

//Declaring arithmetic exception using throws

void sample() throws ArithmeticException{

//Statements

}

You can throw one exception at a time but you can handle multiple exceptions by declaring them using throws keyword.  
For example:  
**Throw:**

void myMethod() {

//Throwing single exception using throw

throw new ArithmeticException("An integer should not be divided by zero!!");

}

..

**Throws:**

//Declaring multiple exceptions using throws

void myMethod() throws ArithmeticException, NullPointerException{

//Statements where exception might occur

}

1. What is the use of .join() method in Java?

Join method in Java **allows one thread to wait until another thread completes its execution**. In simpler words, it means it waits for the other thread to die. It has a void type and throws InterruptedException.

1. How many ways we can thread in Java?

In Java, multithreading a thread can be created by the following two ways:

1. By extending the thread class
2. By implementing a Runnable interface

Some of the useful methods of Thread Class that we are going to use:

* start() - It starts the execution of the thread in a separate path and then calls the run() method.
* run() - It performs the action determined in the run() method.
* isAlive() - It returns true if the execution of the thread is not terminated, else it returns false.
* join() - It waits for the thread to die.
* sleep() - To suspend a thread for a certain period of time.

## Method 1 by extending the Thread class

In java multithreading, a thread can be created by extending the java.lang.Thread class. We first create a class that extends Thread and overrides the run() method. Now thread can be created by creating the object of our newly created class and calling the run() method.

Let us take an example of a method that finds the given target element from the array. As the array is not sorted the only efficient way is to traverse the array. Usually, it takes O(n) time to traverse the array where n is the number of elements in the array. Now, we can use multiple threads to reduce the time complexity to find the element in the array.

## Method 2 by Implementing The Runnable Interface (add example)

The another way in java multithreading a thread can be created by implementing the Runnable interface present in java.lang. It is one of the standard Java interfaces that come with the Java platform. It also has a run() method, which is to be overridden by our class.

To create a thread using the Runnable interface,

* our thread class must implement the runnable interface
* the thread class must override the run() method
* the object of the java.lang.Thread class say obj1 and the object of our thread class say obj2.
* passing the obj2 as a parameter to obj1.

## Thread Class vs Runnable Interface

| **Thread Class** | **Runnable Interface** |
| --- | --- |
| It is a class and in Java, a class can only extend one parent class i.e. it doesn't support multiple inheritances. So, we cannot extend any other classes. | It is an interface thus the class can implement multiple interfaces and also extend a parent class as well. |
| We cannot reuse the object of the thread. For each new operation, a new object of the class is required. Thus, it uses more memory. | It is possible to reuse the same object to execute multiple threads by implementing a runnable interface. It requires less amount of memory space. |
| It is inheritance. | It provides composition i.e. we are not modifying or specifying the behavior of the threads. We are just asking it to run something. |
| It has multiple methods that are used to perform actions on the thread. | It has only an abstract method run(). |

1. Which way is better and why?

If you want to implements or extends any other class then Runnable interface is most preferable, otherwise, **if you do not want any other class to extend or implement then Thread class is preferable**. When you extends Thread class, after that you can't extend any other class which you required.

1. Why we need Synchronization in Java?

**Synchronization** in Java can be thought of as a queue at a ticketing counter where a number of customers are waiting for the current customer to finish. As soon as the current customer gets their ticket, all the other customers move one position closer to the ticket counter.

Synchronization deals with threads in the same way as a ticketing counter. **Threads** allow multiple processes to be conducted in multiple ways.

Suppose a process adds and multiplies a particular array of elements – two threads can be used to implement this, one can be used to multiply, and the other can perform addition. This allows less CPU usage and saves a lot of CPU time.

Synchronization, in particular, is a concept that ensures that exactly one thread is entertained at a time (i.e., multiple threads do not function at the same time​).

Synchronization allows a programmer to perform [concurrent programming](https://www.educative.io/edpresso/what-is-concurrent-programming) and prevents data corruption.

It stop thread to interfering with object data at the `same time

1. What if we Synchronize method instead of block?

|  |  |
| --- | --- |
| Method | block |
| If multiple threads try to access a method, then the thread that would come first will get the lock and perform its execution. And the rest of the thread will wait for the first thread to finish its execution | Synchronized block has limited scope than synchronized method. |
| A synchronized method assigns an object-level or class-level corresponding lock | synchronized block assign a lock to the object based on the parameter. |
| A synchronized method lock on the entire functionality of the method concerned. | synchronized block is used to acquire the lock on a small number of the consecutive statement. (i.e Critical section area) |
|  |  |

1. Do you know ThreadPool in Java?

A thread pool in [Java](https://www.java.com/en/) reuses the previously created threads to execute new requests. This also solves the problem of thread cycle overhead and running out of resources. Since the threads already exist, when a new request arrives, the delay due to the creation of a new thread is now eliminated, making the application significantly more responsive. Additionally, using Java thread pools can make it easier to control the number of threads that are active at a time.

1. What is the use of Threadpool in Java?

A thread pool in Java **reuses the previously created threads to execute new requests**. This also solves the problem of thread cycle overhead and running out of resources.

1. How can we make sure Thread is still going on or not?

The isAlive() method is used to check if a thread has finished executing. If the thread is still running, it returns true, otherwise false. If we use isAlive method before the start method, then it will print false but after the start method, it will print true.

1. Difference between Process and Thread?

|  |  |  |
| --- | --- | --- |
| **Comparison Basis** | **Process** | **Thread** |
| Definition | A process is a program under execution i.e. an active program. | A thread is a lightweight process that can be managed independently by a scheduler |
| Context switching time | Processes require more time for context switching as they are heavier. | Threads require less time for context switching as they are lighter than processes. |
| Memory Sharing | Processes are totally independent and don’t share memory. | A thread may share some memory with its peer threads. |
| Communication | Communication between processes requires more time than between threads. | Communication between threads requires less time than between processes. |
| Blocked | If a process gets blocked, remaining processes can continue execution. | If a user level thread gets blocked, all of its peer threads also get blocked. |
| Resource Consumption | Processes require more resources than threads. | Threads generally need less resources than processes. |
| Dependency | Individual processes are independent of each other. | Threads are parts of a process and so are dependent. |
| Data and Code sharing | Processes have independent data and code segments. | A thread shares the data segment, code segment, files etc. with its peer threads. |
| Treatment by OS | All the different processes are treated separately by the operating system. | All user level peer threads are treated as a single task by the operating system. |
| Time for creation | Processes require more time for creation. | Threads require less time for creation. |
| Time for termination | Processes require more time for termination. | Threads require less time for termination. |

1. Do you know Collections framework in Java?

Collections are like containers that group multiple items in a single unit

1. Can you name any collections which you are using?

**ArrayList, LinkedList, HashMap, TreeMap, HashSet, and TreeSet**.

1. Do you know How HashMap works internally?

Hash code for null value is always zero. So index is also zero in null value

A hashmap uses a hashtable, however, it is internally implemented using two data structures namely an array and a linked list. Whenever you declare a hashmap, internally, it will create an array of buckets. The buckets are referred to as nodes or you can say a linked list. A node can represent :

* Hashcode
* Key
* Value
* Address of the next node

Diagram, text

Description automatically generated

Now, when you insert values in a key using put() method of the HashMap class, the hashcode will be generated by the put() method. Hashcode makes the process of receiving values faster and easier. And this hashcode is further computed and it will generate the index for storing the value.

**Collision in a HashMap :**

A Hash Collision is a situation where for two or more distinct records, the hash function returns the same bucket position in the HashTable. This is the main disadvantage of using hashmap.

HashMap uses these two methods to handle collision:

1. Separate Chaining
2. Open Addressing

Data Structures used for Separate Chaining are -

1. Linked List
2. Self Balancing Binary Search Tree
3. Dynamic Sized Array

HashMap replaces linked list with a binary tree when the number of elements in a bucket reaches certain threshold.

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

https://levelup.gitconnected.com/how-does-hashmap-work-internally-612dd950822f

1. How set works internally?

 It does not allow duplicate elements.

 When we try to pass the same element that is already available in the Set, then it will not store in the Set.

 It is used to model the mathematical set abstraction.

 You can use it when you want data/Elements with unique item

The elements you add into HashSet are stored as keys of this HashMap object.

 It does not guarantee the order of elements.

 It is achieved by storing elements as keys with the same value always.

 It constructs a collection that uses a hash table for storing elements.

 It contains unique elements.

 It inherits the AbstractSet class.

 It also implements the Set interface.

 It uses a technique to store elements is called hashing.

 HashSet uses [**HashMap**](http://www.9mood.com/how-hashmap-works-internally-java/)internally in Java.

The elements you add into HashSet are stored as keys of this HashMap object.

**HashSet internally uses HashMap to store it's elements**. Whenever you create a HashSet object, one HashMap object associated with it is also created. This HashMap object is used to store the elements you enter in the HashSet. The elements you add into HashSet are stored as keys of this HashMap object.

 **[Important Iterator]** HashSet does not have any method to retrieve the object from the HashSet. There is only a way to get objects from the HashSet via **Iterator**.

 HashSet has default initial **capacity** = **16**.

 HashSet has default **loadfactor** = **0.75** or **75%**

 The capacity may increase automatically when more elements to be store

S

Hashset use the hash map internally and when the object is passed it create a key so the object is passed in the key but value is created by it self like a dummy value.

1. Difference between Array and Arraylist, HashSet and HashMap, Arraylist and Linkedlist, HashSet and TreeSet?

|  |  |
| --- | --- |
| Array | ArrayList |
| Length of Array is static that means one cannot change its length that has been already defined by the developer to that particular element. This Array needs to specify the size of the elements. | ArrayList uses the size() method to compute the size of the elements. Also, it is dynamic, which means one can change the size of the arraylist if the elements are modified in it, which means the length of the arraylist is variable. |
| In other words, the length of the elements in the array is static or  requires more memory to store the elements and less time to iterate the elements. | ArrayList requires more memory to store the elements as well as more time to iterate. |
|  Array does not allow generics, though multidimensional in nature. |  ArraList enables the use of generic and single dimensional in nature. |
|  In the array, giving references to objects or elements  depends upon the type of array such as [primitive type](https://www.simplilearn.com/tutorials/java-tutorial/data-types-in-java) or object type. |  In arrayList, we can convert the primitive int data type into an Integer  Arraylist need to convert int primitive data type to its wrapper class integer |
| Cannot remove the element of the array as size is fixed it need to be replaced with other variable | Can remove the element as the size is dynamic |

|  |  |
| --- | --- |
| Hashmap | Hashset |
| Hashmap is the implementation of Map interface. | Hashset on other hand is the implementation of set interface. |
| Hashmap internally do not implements hashset or any set for its implementation. | Hashset internally uses Hashmap for its implementation. |
| HashMap Stores elements in form of key-value pair i.e each element has its corresponding key which is required for its retrieval during iteration. | HashSet stores only objects no such key value pairs maintained. |
| Put method of hash map is used to add element in hashmap. | On other hand add method of hashset is used to add element in hashset. |
| Hashmap due to its unique key is faster in retrieval of element during its iteration. | HashSet is completely based on object so compared to hashmap is slower. |
| Single null key and any number of null value can be inserted in hashmap without any restriction. | On other hand Hashset allows only one null value in its collection,after which no null value is allowed to be added. |
| Yes duplicates values are allowed but no duplicate key is allowed | Duplicate keys are not allowed |

|  |  |
| --- | --- |
| Arraylist | Linkedlist |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

|  |  |
| --- | --- |
| Hashset | Treeset |
| Hash set is implemented using HashTable | The tree set is implemented using a tree structure. |
| HashSet allows a null object | The tree set does not allow the null object. It throws the null pointer exception. |
| Hash set use equals method to compare two objects | Tree set use compare method for comparing two objects. |
| HashSet does not maintain any order | TreeSet maintains an object in sorted order |
| Hashset is faster | Treeset is slower |
| Requires less memory | Requires more memory |

1. Default capacity of HashMap in Java and How it will increase its size?

The initial capacity is the capacity of an HashMap at the time of its creation. The default initial capacity of the HashMap is 24 i.e 16.

**As the number of elements in the HashMap increases, the capacity is expanded**. The load factor is the measure that decides when to increase the capacity of the Map. The default load factor is 75% of the capacity. The threshold of a HashMap is approximately the product of current capacity and load factor.

1. Constructor rules in Java?

1.Name of the class and name of the constructor must be same

Class test(){

Test(){

}

}

2. if wed give return type to constructor the compiler will treat as a method

Eg”

Class test() {

(return type is void so it is method it is no more constructor)Void class(){

}

}

3 modifier for the constructor that are allowed are public, default, protected, private

1. What is the use of this, super keyword in Java?

Super keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Super refers to parent call only

This keyword

This keyword refers to the class it is in 

This keyword does not work with anything that is static

1. Various cases of super and this method

The first line in the constructor is this or super keyword if we don’t place it it will automatically consider as super keyword

We cannot take super keyword in second line. Call to super must be in the first line in the constructor

Class test{

Test() {

Super();

Sop(asd);

}

}

We cannot use the super and this together as both need to be in the first line in constructor. So we need to use anyone from the super or this.

We can use super or this only in constructor we cannot use it in the method.

1. Do you know Java 8 features?

forEach() method in Iterable interface

default and static methods in Interfaces

Functional Interfaces and Lambda Expressions

Java Stream API for Bulk Data Operations on Collections

Java Time API

Collection API improvements

Concurrency API improvements

Java IO improvements

<https://www.digitalocean.com/community/tutorials/java-8-features-with-examples>

1. Can you explain me what is Lambda in Java 8?

Interface:

We can not create the object of the interface.

To create a object of the interface we need to create a class which implements the interface and with that interface we are able to create the object of that implemented class.

Functional interface:

If you have one abstract method is called a functional interface

If it has more than one method in interface than it is not functional interface

This are also called sam interface because they have single abstract method.

Functional interface allowed the method that are in object class. Every class in java extends the object class.

Create the object of interface which will give error but we can create a class after creating the object like

Interface A {

Void show();

}

Public class Demo {

Public static void main(String[] args){

A obj = new A() {

//Anonymous inner class

Public void show(){

System.out.println(“hi”);

}

}

}

}

Lambda:

In lambda you are only able to define the implementation of one method . to define more than one method in lambda we need to create an anonymous class.

Lambda method can directly call the interface without implementing the interface and create the object of the implemented class.

Interface A {

Void show();

}

Public class Demo {

Public static void main(String[] args){

A obj = () ->{

System.out.println(“hi in show)

}

}

}

1. Do you know Stream API in Java 8?

Stream API is used to process data

Two ways to create a stream using Of() method and Stream method

1. Of method

If you want to give elements to the stream than we can use of method

Stream s1 = Stream of(1,2,3,4,5,6)

1. Stream method

If you already have a data in collections using that collection you can create a stream using a stream method

Stream cannot be reused. Once we consumed the stream value, we cannot use the the stream one more time

You can use multiple stream methods in one line

Eg:

List<integer> numbs = arrays.asList(6,5,23,5,6);

Numbs.stream()

.sorted()

.map(n -> n\*2)

.foreach ( n-> s.o.p(n));

ArrayList al = new ArrayList<>();

al.add(10);

Stream s1 = new Stream();

Operations that we can perform on stream methods are:

Intermediate

* 1. Filter()
  2. Map(), Flatmap()
  3. Skip(), limit(), distinct()

Will perform the operations ad give a new stream as a output

Terminal

1. Count()
2. Collect()

Terminal operations gives us a result not the new stream

* Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
* Stream is functional in nature. Operations performed on a stream does not modify it's source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
* Stream is lazy and evaluates code only when required.
* The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

It does not store data, it is used to perform bulk operations, reduce bulk operations it is used on collections.

1. Programs that we can achieve using Collections.

What are the uses of collections?

Collections are used **to store, retrieve, manipulate, and communicate aggregate data**.

1. Why String is Mutable in Java?

String is immutable, so its value cannot be changed. Strings are not mutable

If string is not immutable, any hacker can cause a security issue in t

1. Difference between String, StringBuilder and StringBuffer?

|  |  |  |
| --- | --- | --- |
| String | StringBuffer | StringBuilder |
| Stored on heap memory or string constant pool | Stored on heap | Stored on heap |
| String Is immutable. Value cannot be modified | Mutable | Mutable |
| Synchronized, multiple thread can occur simultaneously | Synchronized, multiple thread can occur simultaneously | Unsynchronized or non-synchronized, multiple thread can access the method at a time |
| Thread safe | Thread safe | Not thread safe |
| Less efficient as it create a new object | Efficient | More efficient there no locking and so it is facter |

String :

String can be literal or Object

StringBuffer and StringBuilder

* Classes are used to perform operations on the string

StringBuffer sbf = new StringBuffer(“hello”);

Sbf.append(“world”);

StringBuilder sbd = new StringBuilder(“herllo”);

Sbd.append(“World);

1. Do you know ConcurrentHashMap in Java?

<https://www.youtube.com/watch?v=_A0Wty5Aeis>

Concurrent hashmap can use multiple thread at a same time. Lets say is the value which we are entering in the the node and id the address they go into are different than the all the values that have different values will be entered on the node at the same time but if the address is the same where value need to go than it is going to wait for the first thread to complete and than it will enter the other value at that address.

1. How thread communicates with each other?

Multi threading in java

* Multithreading in java is a process on executing threads simultaneously.
* A thread id a lightweight sub-process, the smallest unit of processing.

Eg:

Blueprint of runnable thread

Class MyThread implements runnable {

Public void run(){

//task

}

}

MyThread t = new MyThread

Thread thread = new thread(t)

Thread.start();

Blurprint of thread class

Class MyThread etends Thread

{

Public void run(){

//task

}

}

MyThread t = new MyThread();

t.start();

Thread operations

Thread class provides methods to perform operations with threads.

This thread class is present in java.lang package so we don’t need to import this package.

Public string getName()-returns the name of thread

Public void setName(String name)-set the name of thread.

Public void run()- contain the task of thread

Public void start()- start thread by allocating resources.

Public long getId()- returns the id of thread

setPriority(p), getPriority()- set and get the priority.

Sleep(),joim(),interrupt(),resume(),stop()

Daemon Thread

Daemon thread in java is a service provider thread that provides services to the user thread

Set daemon(boolean)

Public Boolean isDaemon()

Garbase Collector is a best example of Daemon Thread

Producer consumer problem

Synchromnized, Inter-Thread communications

1. Explain Java 8 features with an Example.

https://www.digitalocean.com/community/tutorials/java-8-features-with-examples

1. Do you know frameworks of Java?

Spring boot, spring, react.js, apache spark, node.js, angular, hibernate

1. Which framework you have used so far and why?

Spring boot and mvc, spring

Spring Boot **helps create apps that aren't tied to a specific platform and that can run locally on a device without an internet connection or other installed services to be functional**.

1. Difference between Spring and Spring-Boot framework?

|  |  |
| --- | --- |
| Spring | Spring boot |
| Dependency injection framework | A suite pre-configured frameworks and technologies  That is used to remove boiler plate configuration |
| Manage lifecycle of java classes | The shortedt way to have a spring application up and running |
| Boiler plate configuration  Write a lot of code | Add Annotations to create application of the spring |
| Takes time to have a spring application up and running |  |

1. Features of Spring Framework.
2. Do you know Dependency Injection in Java?
3. Explain me DI and IoC in Spring.
4. What is AOP in Spring?
5. Explain what is tight and loose coupling in Java?
6. How Spring loads its application and what is happening behind the scenes.
7. What are the scopes of Beans with an example in Spring?
8. Name me annotations which you have used so far.
9. Explain annotations with an example in Spring.
10. Do you know what is REST API?
11. How you implement REST API in Java?
12. Difference between @Component and @Controller , @Component and @Service , @Component and @Repository, @Service and @Repository?
13. Do you know @Transactional in Spring?
14. What are the possible ways to create Transactions in Java and How it will work?
15. How Transaction Manager works internally?
16. How JWT token works?
17. Do you know any design Patterns in Java?
18. Name me some of the design patterns that you are aware of and which you are using in your project?
19. Explain Factory Pattern in Java.
20. Explain me Singleton and Mediator Pattern in Java.
21. Do you know Unit Testing in Java?
22. What kind of tests you are doing in your Project?
23. Do you know difference between Unit and Integration testing?
24. Do you know Mockito framework in Java?
25. Can you please explain me @Mock and @InjectMock?
26. How you write down Unit TestCase for method with example?
27. Can we test Private Methods using Junit?
28. Do you know branching strategy your Project is using?
29. Explain different Strategy and its usages.
30. Difference between Pull and Fetch, Commit and Push, Rebash and Merge?
31. How to cherry pick commit in Git?
32. Do you know difference between Monolithic and Microservices architecture?
33. Advantages and Disadvantages of Microservices?
34. How microservices communicates to each other?
35. How you can deploy and Project in EC2?
36. What is the difference between Docker and Kubernetes?
37. How do you deal with your issues in Your Project?
38. How many types of Scaling possible?
39. How Load Balancer work?
40. How to design a REST API which will communicate to other services?
41. How are you securing your REST API’s?
42. Difference between Authentication and Authorization?
43. MySQL – Select, Update, Delete, Inner Joins, Left and Right Joins.
44. How to get Max salary of an employee from emp table?
45. Difference between Primary , Foreign , Unique and Candidate Keys?
46. Do you know indexing in MySQL?
47. How to provide index on table?
48. How you will optimize MySQL Query performance?
49. How many types of Query we can use in Spring?
50. Do you know JPA in Spring?
51. Difference between Native and Names Query?
52. How persistence manager works?
53. Do you know Profiles in Spring?
54. Why we need Profiles in Spring?
55. How to Provide multiple Profiles in Spring and How can we use at runtime?
56. Do you know Spring Actuator ?
57. Why Spring Actuator helps in Production?
58. How to achieve IS-A relations in Spring with annotations?
59. Difference between Eager and Lazy loading in Java?
60. How to achieve relations ships between Entities?
61. How many types of Relationships we have?
62. Can you explain me @ManyToOne in Spring?
63. What if Transaction failed?
64. What is the SOLID Principles in Java?
65. What is Aggregation and Composition?
66. How exception handling works for method overloading and overriding?
67. Do you know Anonymous inner class?
68. Collection thread safety - What are the Thread-safe classes in Java Collections framework? How to run thread?
69. Why use concurrent HashMap and how it works internally? what happens if you call run() directly?
70. How many classes needs to be created to implement factory method pattern?
71. Difference between encoding and encryption?
72. Microservice architecture in detail.
73. Why Dependency injection needed? How is it implemented?
74. How you write test cases in Junit?
75. Have you written a unit test + component test + integration test?
76. Overall architecture of current application.
77. **How to make a Class immutable.**
78. Thread Life cycle
79. What is Marker interface? What is the used of it?
80. Comparable and Comparator in Java and Its usage.
81. Different between Bean-Factory and Application-Context
82. What is session in hibernate?
83. What is Session-factory in hibernate?
84. Convince me why you are the best for this position.
85. What are your Strength and Weakness?
86. Can we deploy two applications with single tomcat server?
87. Why is Spring boot used for micro service app?
88. how to configure database properties in spring boot?
89. how to put validation for user details coming in request?
90. How you handle transactions in your application?
91. what if you found a bug in production? how you will use git branching for production bug fix?
92. Do you have any questions for me?