INTERVIEW QUESTION AND ANSWERS

* Oops concept

Ans:- Core concepts of OOPs are :

* Polymorphism
* Abstraction
* Encapsulation
* Inheritance

**What is Abstraction?**

* [Abstraction](https://java2blog.com/abstraction-java-example/) is a concept of showing only important information and hiding its implementation. This is one of the most asked Oops interview questions as it checks basic oops concepts for java programmers.   
  **For example:**   
  When you see a car, you know it is running but how it running internally, you may not aware of it.   
  This is Abstraction. You just expose required details.

**3. What is encapsulation?**

[Encapsulation](https://java2blog.com/encapsulation-java-example/) is process of wrapping data and function into single unit. You can use access modifier for variables, so other classes may not access it directly but it can be accessed only through public methods of the class. You can create class fully encapsulated using private variables.

**4. What is difference between Abstraction and Encapsulation?**

* Abstraction is a concept of showing only important information and hiding its implementation where as Encapsulation provides a barrier to access of data and methods.
* Abstraction is more about design concept and Encapsulation is about implementation.

**Encapsulation in Java can be achieved by:**

1. Declaring the variables of a class as private.
2. Providing public setter and getter methods to modify and view the variables values.

In java, abstraction is achieved by **interfaces and abstract classes**. We can achieve 100% abstraction using interfaces. Abstract classes and Abstract methods : An abstract class is a class that is declared with an abstract keyword.

[Polymorphism](https://java2blog.com/polymorphism-java-example/)

[Polymorphism](https://java2blog.com/polymorphism-java-example/) means one name many forms. It is concept by which you can perform same action in different ways.

There are two type of polymorphism in java.

* Compile time polymorphism
* Run time polymorphism

Compile time polymorphism

* in which a call to an overloading method is resolved at compile time rather than at run time. In this process, we done overloading of methods is called through the reference variable of a class here no need to superclass.   **Method Overloading in Java:**   
  If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

Run time 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.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

You can implement polymorphism using

* [Method overloading](https://java2blog.com/method-overloading-in-java/ “Method overloading”)
* [Method overriding](https://java2blog.com/method-overriding-in-java/

[**Inheritance**](https://java2blog.com/inheritance-java/)

 allows use of properties and methods of another class (Parent class), so you can reuse all methods and properties. Basically we can access an the non static members of parent class to the child class.

**Super keyword:** super keyword is use to access the member of parent class to the child class and Both static and non static members are accessible.

**This keywords:-** this keyword is use to point out the current execution in a method of object Can access only an static and non static members. This keyword can not access an local variable.

Super and this keyword can use together

 You **can use this and super in the same method**, just not as constructor calls.

public Child() {

super(); // OK

this(); // ❌ Error: Constructor call must be the first statement

}

void display() {

System.out.println(this.value); // Refers to Child's value

System.out.println(super.value); // Refers to Parent's value

}

**Can static be overridden.**

Static methods cannot be overridden, because static methods cannot been inherited.

**9.What is multiple inheritance?**

**Answer:**   
When child class can inherit from multiple parent classes. This mechanism is known as multiple inheritance.

**10. What is diamond problem in case of multiple inheritance?**

**Answer:**   
Let’s understand this with the help of simple example.   
Let’s assume:

* Class A has two child classes B and C.
* Class D has two parent classes B and C.
* methodCommon() of A is overriden by classes B and C.
* When you call methodCommon() on instance of D, which method should get called(From class B or C)

Above problem is known as diamond problem in the context of multiple inheritance.

Extends a,b will throw compilation error.

**11. Why java does not support multiple inheritance?**

Java avoided multiple inheritance due to diamond problem and to make it less complex.

Static binding is resolved at compile time. Method overloading is perfect example of static binding. 

Dynamic binding is resolved at run time. Method overridng is perfect exmple of dynamic binding.

**Is java pass by value or pass by reference**

Yes java is a pass by value or the pass by reference

(but let the see screen first)

Copy Constructor

A copy constructor in a Java class is a [constructor](https://www.baeldung.com/java-constructors) that**creates an object using another object of the same Java class**.

That’s helpful when we want to copy a complex object that has several fields, or when we want to make a [deep copy](https://www.baeldung.com/java-deep-copy) of an existing object

Functional interface

Runnable –> This interface only contains the run() method.

Comparable –> This interface only contains the compareTo() method.

ActionListen

er –> This interface only contains the actionPerformed() method.

Callable –> This interface only contains the call() method.

Functional interface are the known as a SAM  as where its contain only one abstract method ,they have only one functionality to exhibit and this interfaces are annotated with @functionalinterface

Incomplete method:

Immutale class

Immutable class means once the object is create of that class , then its conent cann not be change through out the execution.

The class must be declared as final so that child classes can’t be created.

Data members in the class must be declared private so that direct access is not allowed.

Data members in the class must be declared as final so that we can’t change the value of it after object creation.

A parameterized constructor should initialize all the fields

Provide only getter not setter.

**Synchronization ? adv and disadva??**

We use synchronization to control the execution of threads.The basic requirement of Synchronization is all the threads must be set on same

If multiple threads are operating on the same Java object, then maybe the chance of data inconsistency problem, this is called race condition. we can overcome this problem by synchronized keyword.

We add synchronized keyword before run method.

**What is stream api??**

Stream API is a newly added feature to the Collections API in Java Eight. A stream represents a sequence of elements and supports different operations (Filter, Sort, Map, and Collect) from a collection.

**Languages.stream().forEach(System.out::println)**

**Thread? Multithreading?**

**A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.**

**Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.**

**Multithreading in Java is a process of executing multiple threads simultaneously.**

**Use of executor services?**

**ExecutorService automatically provides a pool of threads and an API for assigning tasks to it.**

Executor service framework

Java provides its own multi-threading framework called the Java Executor Framework.

The[java.util.concurrent.Executors](https://www.geeksforgeeks.org/java-util-concurrent-executor-interface-with-examples/) provide factory methods that are being used to create [ThreadPools](https://www.geeksforgeeks.org/thread-pools-java/) of worker threads.

**Some types of Java Executors are listed below:**

1. SingleThreadExecutor
2. FixedThreadPool(n)+
3. CachedThreadPool
4. ScheduledExecutor

Let us discuss these popular java executors to some details what exactly they do to get a better idea prior to implementing the same.

**Executor 1:** SingleThreadExecutor

A thread pool of single thread can be obtained by calling the static [*newSingleThreadExecutor()*](https://www.geeksforgeeks.org/difference-between-executorservice-execute-and-submit-method-in-java/) method of the Executors class. It is used to execute tasks sequentially.

**Syntax:**

ExecutorService executor = Executors.newSingleThreadExecutor();

**Executor 2:** FixedThreadPool(n)

As the name indicates, it is a thread pool of a fixed number of threads. The tasks submitted to the executor are executed by the n threads

ExecutorService fixedPool = Executors.newFixedThreadPool(2);

**Executor 3:** CachedThreadPool

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. Calls to execute will reuse previously constructed threads if available. If no existing thread is available, a new thread will be created and added to the pool.

ExecutorService executorService = Executors.newCachedThreadPool();

**Executor 4:**ScheduledExecutor

This executor is used when we have a task that needs to be run at regular intervals or if we wish to delay a certain task.

ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1);

**Clonable interface?**

**The Java.lang.Cloneable interface is a marker interface. This interface allows the implementing class to have its objects to be cloned instead of using a new operator.**

**is overriding and overloading allowed in constructor in java**

**It is never possible. Constructor Overriding is never possible in Java. This is because, Constructor looks like a method but name should be as class name and no return value.**

**Yes! Java supports constructor overloading.**

**Static variable: the variable belongs to the class itself rather than to any specific instance of the class.**

**Classsname.var\_name (how to use)**

**Wrapper class?**

**The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.**

**Type = int**

**Class = Integer**

**TRANSIENT KEYWORD**

  The transient keyword in Java is used to avoid serialization.

If any object of a data structure is defined as a transient ,

 then it will not be serialized.

its basically used to hide the information that is being showed...

**To create an immutable class in Java, you have to do the following steps.**

create final class.

set properties only using only constructor.

Don’t provide setter methods for variable.

Make all mutable fields final so that its value can be assigned only once.

**String are immutable?**

 Strings are immutable precisely so that their references can be treated as a normal variable and

one can pass them around,

 between methods and across threads, without worrying about whether the actual String object it's pointing to will change.

why strings are immetable ?

Strings in Java are **immutable**—meaning once a String object is created, its value cannot be changed. This design choice is intentional and brings several powerful benefits:

🔐 1. **Security**

Strings are often used to store sensitive data like usernames, passwords, and file paths. If they were mutable, malicious code could alter them mid-operation, leading to security vulnerabilities.

🧵 2. **Thread Safety**

Since strings can't be changed, multiple threads can safely share the same String instance without needing synchronization. This makes concurrent programming simpler and safer.

💾 3. **String Pooling**

Java uses a **String pool** to store string literals. When you write:

String a = "hello"; String b = "hello";

Both a and b point to the same object in memory. This is only possible because strings are immutable—otherwise, changing a would also affect b.

⚙️ 4. **Caching and Performance**

Immutable strings can be cached and reused without worrying about unexpected changes. This improves performance, especially in operations like hashing (e.g., when used as keys in HashMap).

🧪 Example

String s1 = "Java"; String s2 = s1; s1 = s1.concat(" Rocks"); System.out.println(s1); // Java Rocks System.out.println(s2); // Java

Even after s1 is modified, s2 remains unchanged—proof that the original string wasn't altered, just a new one was created.

**Serialization**

Converting an object into byte stream is known as Serialization.

The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object.

**DIFFERENCEE BETWEEN == AND EQUALTO**

== checks if both the objects points to the same memory location while equal() checks the comparison of values inside object

**Failsafe and Failfast**

Failfast iterator throws an concurrentmodificationexception is a collection is modified while iterating over it. It uses original iterator for the transversal over iterator, they are memory savers, will returned by arraylist , vector and hashmap

Failsafe , we can perform modification while iterating , it wwill not throw exception.it uses copy of collection for traversal over element

**Comparable and Comparator**

Comparable and comparator are both the interfaces are used to sort collection element

 Comparable provide single sorting sequence ,means sorting will on the basis of single element such as id , name

                        Comparator provide multiple sorting sequence ,means sorting will on the basis of multiple element such as id , name

Comparable affects original class

                            Comparator doesn’t affect original class, it does not modifies the original class.

Comparable provides compare to method

                                   While comparator provides compare methods

Comparable present inside into java.lang,

                      Comparator present inside java.util

Throw throws throwable

·  throw: Used to explicitly throw an exception.

·  throws: Used in method signatures to declare the exceptions that a method can throw.

·  Throwable: The superclass of all errors and exceptions.

Final finally finalize:

·  final: Used to declare constants, prevent method overriding, and prevent inheritance.

·  finally: Used in exception handling to execute code regardless of whether an exception is thrown or not.

·  finalize: Used to perform cleanup before an object is garbage collected, but it is deprecated and should not be relied upon for resource management.

Garage collector

Its primary purpose is to identify and discard objects that are no longer needed by a Java application, thus freeing up memory resources for reuse.

Static binding : method overloading early binding

Dynamic binding : method overriding , late binding

Ternary operator : its concise way of implementing if else block ., it takes condition ,  value if true , value if false

Where do you use serialization:session management in web apps and file handeling mechanism

**String buffer and string builder:**

String buffer is synchronized in nature

String builder is non synchronized in nature

String buffer is thread safe

While String builder is not  a thread safe

String  buffer is  less efficient

 While String builder is more efficient

String buffer is introduced In java

**While it is introduced in**

**String Tokenizer:**

The String tokenizer class allows us to break  a string into a small tokens.its a legacy class of java.its a one of the simple way to beak a string

  What is the String pool in Java? What is the difference in String pool between Java 6 and 7? ([answer](http://javarevisited.blogspot.sg/2016/07/difference-in-string-pool-between-java6-java7.html))   
It's a pool of cached String objects for minimizing the number of String instances and improving performance by sharing the same instance with multiple clients and reducing garbage collection. Prior to Java 7, the String pool was located on meta-space where class metadata was stored but from JDK 7 onwards it's moved into heap space.   
 

1. **The difference between String and StringBuffer in Java?**([answer](http://www.java67.com/2016/10/5-difference-between-stringbuffer.html))   
   The key difference between String and StringBuffer is that String is Immutable while StringBuffer is mutable, which means you can change the content of a StringBuffer without creating separate String objects.   
      
   The difference between StringBuffer and StringBuilder in Java?

Now, this is interesting because both StringBuffer and StringBuilder represent mutable String. This is also asked a follow-up question of the previous question.   
   
Anyway, the real difference is that methods of StringBuffer like append() are synchronized, hence slow, while those of StringBuilder are not synchronized, hence fast   
   
**20) Is String thread-safe in Java? Why?** ([answer](http://www.java67.com/2014/01/why-string-class-has-made-immutable-or-final-java.html))   
Yes, String is a thread-safe because it's Immutable. All Immutable objects are thread-safe because once they are created, they can't be modified, hence no issue with respect to multiple threads accessing them.   
   
**5) Can you convert user tread into daemon thread and vice-versa? Explain with example?**

Yes, you can convert user thread into daemon thread and vice-versa using setDaemon() method. But, it has to be done before starting the thread

**14) Does the thread releases the lock it holds when it is going for sleep?**

No. When the thread is going for sleep, it does not release the synchronized locks it holds.

**18) I want only some part of the method to be synchronized, not the whole method? How do you achieve that?**

This can be done using synchronized blocks

The **Java String class** **intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from memory if it is created by a new keyword. It creates an exact copy of the heap string object in the String Constant Pool.

1. String str = **new** String("Welcome to JavaTpoint.");
2. String str1 = **new** String("Welcome to JavaTpoint");
3. System.out.println(str1 == str); // prints false

The println statement prints false because separate memory is allocated for each string literal. Thus, two new string objects are created in the memory i.e. str and str1. that holds different references.

In order to put the strings in the string pool, one needs to call the **intern()** method. Before creating an object in the string pool, the JVM checks whether the string is already present in the pool or not. If the string is present, its reference is returned.

1. String str = **new** String("Welcome to JavaTpoint").intern(); // statement - 1
2. String str1 = **new** String("Welcome to JavaTpoint").intern(); // statement - 2
3. System.out.println(str1 == str); // prints true

In the above code snippet, the intern() method is invoked on the String objects. Therefore, the memory is allocated in the SCP. For the second statement, no new string object is created as the content of str and str1 are the same. Therefore, the reference of the object created in the first statement is returned for str1. Thus, str and str1 both point to the same memory. Hence, the print statement prints true.

1. String s3=s1.intern();//returns string from pool, now it will be same as s2

**Thread?**

Multi tasking done at programming level.

Two ways to create threads.

* Using threads class
* Using runnable interface

**Using threads class**:-run method is used to run the thread .here we inherit the run  method and override it.to start a run method , we used  start method of thread class.

Run thread is an user defined thread while main method is by-default an thread. and always the main threads runs first then remaining threads run. To use the thread , first extends the threads, then override the run method.

**Using runnable interface:-**steps to create threads using runnable interface

* Inherit the runnable interface using implements keyword.
* Then create variable of thread using new keyword.
* Then use t1.start(start method).

**Thread synchronization**

When two threads   are working on a same data , in non synchronized manner ,then the data will get corrupted. The synchronization is necessary for reliable communication between threads.

**Synchronization**

If a method is made synchronized then a thread which acquires a lock will only execute.

 Every object have one lock which will he release the lock only after the complete execution.

And until then the rest threads will wait for the lock to be released.as like that , we can prevent the data corruption.

**Wait , notify , notify all**

Wait:-wait is used to hault the execution of thread.

Notify :- notify is used to resume the execution ,of anyone thread.

Notify all :- will used to end the hault of all threads

**Thread life cycle**

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

**Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time.

**Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Timed Waiting:** here the threads lies in waiting state for  the specific time , not forever.after the time overs the threads wakes up and start its execution where its left earlier.

**Terminated:** the thread reaches in this state only from two reasons:

1.when thread is finished its job

2. when unhandled exception occurs .

Deadlock:-when two threads are waiting for each other to releasing a object lock and neither anyone thread is releasing a lock, this state called a deadlock.

**Resolution strategies include:**

* **Avoid Nested Locks:** Avoid holding multiple locks at the same time.
* **Use a Timeout:** Use tryLock method with a timeout to acquire locks.
* **Lock Ordering:** Ensure that all threads acquire the locks in the same order.

**Sleep:- this method is used to** hault the current execution of the thread for the specification time duration.

**Wait:-** The Wait() method is responsible for sending the calling thread into the waiting state. The Thread remains in the waiting state until another thread doesn't invoke the [notify()](https://www.javatpoint.com/java-thread-notify-method) or [notifyAll() method](https://www.javatpoint.com/java-thread-notifyall-method) for that object.

**Thread pool:**-thread  pool is a collection of threads.when a request being handled by using thread ,then we perform the operation and put the thread back in thread pool.we didn’t create and destroy the thread every time , it will reduce the efficiency.rather after the thread usage , the thread will stored back to the pool for the further  use.

Join:-allows one thread to wait until another thread complete its execution

**Thread Priority:-** Thread class defines the three types of priorities

**Min:1, normal:5 , max:10**

**Runnable and Callable:**

 Runnable present inside java.lang

                     While callable is present inside java.util.concurrent

Runnable cannot throw checked exception

          While callable throws checked Exception

In runnable , one needs to override run method

       In callable , one needs to override the call method

**Thread Scheduler:-**

It’s the component of jvm which decides the execution order of multiple threads.it decides the order in which thread should run.

It basically used two technique

* **Preemptive scheduling**
* **Time sliced scheduling**

**Preemptive scheduling:** here scheduling based on priority

**Time sliced scheduling:**where , every running thread should run for specific fixed time period.its non-priority scheduling

What is the difference between start() and run() methods in Java threads?

**Answer:**

start() **method:** This method is used to start a new thread. The start() method calls the run() method internally, which allows the thread to execute concurrently.

run() **method:** This method contains the code that is executed by the thread. If you call the run() method directly, it will not start a new thread but will execute in the current thread.

start() creates a new thread and calls run(). run() executes in the current thread.

What is the difference between wait() and sleep() in Java?

**Answer:**

wait() **method:** This method is used to make the current thread wait until another thread invokes the notify() or notifyAll() methods for that object. It releases the lock held by the thread

sleep() **method:** This method is used to pause the execution of the current thread for a specified period. It does not release the lock held by the thread.

**3) How many types of threads are there in Java? Explain?**

There are two types of threads in Java. They are,

1. User Threads
2. Daemon Threads

User threads are threads which are created by the application or user. They are high priority threads. JVM will not exit until all user threads finish their execution. JVM wait for user threads to finish their task. These threads are foreground threads.

Daemon threads are threads which are mostly created by the JVM. These threads always run in background. These threads are used to perform some background tasks like garbage collection. These threads are less priority threads. JVM will not wait for these threads to finish their execution. JVM will exit as soon as all user threads finish their execution.

join()

The join() method in Java is used to pause the execution of the current thread until the thread on which join() was called has finished executing. This is useful when you need to ensure that a particular thread has completed its task before proceeding.

However, the term "detaching" can be interpreted as allowing a thread to run independently without the main thread waiting for its completion. This is simply achieved by not calling join() on the thread.

Which is better thread class or runnable interface?

Simply put, we generally encourage the use of *Runnable* over *Thread*:

* When extending the *Thread* class, we’re not overriding any of its methods. Instead, we override the method of *Runnable (*which *Thread* happens to implement*)*.
* After extending the *Thread* class, we can’t extend any other class
* From Java 8 onwards, *Runnables* can be represented as lambda expressions

HOW THREAD POOL INTERNALLY WORKS

**Workflow of ThreadPool Execution**

1. Thread pool is created using executerService framework
2. A task is submitted via execute() or submit().
3. If the number of running threads is **less than the core size**, a new thread is created to handle the task.
4. If core threads are busy, the task is placed in the **task queue**.
5. If the queue is full and the number of running threads is **less than the max size**, a new thread is created.
6. If the pool is at max size and the queue is full, the **rejection policy** determines the action (e.g., throwing an exception or discarding the task).
7. Once a thread finishes executing a task, it fetches the next task from the queue.
8. If a thread remains idle for a specified **keep-alive time** and the pool size exceeds the core size, it is terminated.
9. Once our task is done , we need to terminate the thradpool to releases the resources busing terminate method.

Memory in java

1. Class(Method) Area
2. Heap
3. Stack
4. Program Counter Register
5. Native Method Stack

1. Class (Method) Area

The class method area is the memory block that stores the class code, variable code(static variable, runtime constant), method code, and the constructor of a Java program. (Here method means the function which is written inside the class). It stores class-level data of every class such as the runtime constant pool, field and method data, the code for methods.

2. Heap

The Heap area is the memory block where objects are created or objects are stored. Heap memory allocates memory for class interfaces and arrays (an array is an object). It is used to allocate memory to objects at run time

3. Stack

Each thread has a private JVM stack, created at the same time as the thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.

Java Stack stores frames and a new frame is created each time at every invocation of the method. A frame is destroyed when its method invocation completes

4. Program Counter Register:

Each JVM thread that carries out the task of a specific method has a program counter register associated with it. The non-native method has a PC that stores the address of the available JVM instruction whereas, in a native method, the value of the program counter is undefined. PC register is capable of storing the return address or a native pointer on some specific platform.

5. Native method Stacks:

Also called C stacks, native method stacks are not written in Java language. This memory is allocated for each thread when it’s created And it can be of a fixed or dynamic nature.

Feeling lost in the vast world of Backend Development? It's time for a change! Join our [Java Backend Development - Live Course](https://www.geeksforgeeks.org/courses/Java-backend-live?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) and embark on an exciting journey to master backend development efficiently and on schedule.

Types of Access specifiers/ Modifiers in Java

There are four types of access modifiers available in Java:

1. Default – No keyword required
2. Private
3. Protected
4. Public

1. Default Access Modifier

When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default. The data members, classes, or methods that are not declared using any access modifiers i.e. having default access modifiers are accessible **only within the same package**.

2. . Private Access Modifier

The private access modifier is specified using the keyword **private**. The methods or data members declared as private are accessible only **within the class** in which they are declared.

3. Protected Access Modifier

The protected access modifier is specified using the keyword **protected**.

The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

**4.Public Access modifier**

The public access modifier is specified using the keyword **public**.

* The public access modifier has the **widest scope** among all other access modifiers.
* Classes, methods, or data members that are declared as public are **accessible from everywhere** in the program. There is no restriction on the scope of public data members.

JAVA 8 FEATURES

Java 8 provides following features for Java Programming:

* Lambda expressions,
* Method references,
* Functional interfaces,
* Stream API,
* Default methods,
* Static methods in interface,
* Optional class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,

Lambda expressions:-

                                    Lambda expression helps us to write our code in functional style. It provides a clear and concise way to implement SAM interface(Single Abstract Method) by using an expression. It is very useful in collection library in which it helps to iterate, filter and extract data.

**Method References**

Java 8 Method reference is used to refer method of functional interface . It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference.

**Functional Interface**

 An Interface that contains only one abstract method is known as functional interface. It can have any number of default and static methods. It can also declare methods of object class.

Functional interfaces are also known as Single Abstract Method Interfaces (SAM Interfaces).

Use of functional interface?

Functional Interface are use for **lamda expression**. Runnable , Callable , Comparable , Comparator are few examples of Functional Interface.

**Optional**

Java introduced a new class Optional in Java 8. It is a public final class which is used to deal with NullPointerException in Java application. We must import *java.util* package to use this class. It provides methods to check the presence of value for particular variable.

**its use** :  is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as 'available' or 'not available' instead of checking null values.

Shape

**forEach**

Java provides a new method forEach() to iterate the elements. It is defined in Iterable and Stream interfaces.

**Date/Time API**

Java has introduced a new Date and Time API since Java 8. The java.time package contains Java 8 Date and Time classes.

**Default Methods**

Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default keyword are known as default methods. These methods are non-abstract methods and can have method body.

**Java StringJoiner**

Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by a delimiter. Now, you can create string by passing delimiters like comma(,), hyphen(-) etc. You can also pass prefix and suffix to the char sequence.

**Stream api**

Its used to process the collections object. It’s a sequence of object to supports various methods which can be pipelined to produce the desired results.

**//////////////////////java 8 feature end///////////////////////////////////**

**Volatile Keyword**

It is used to modify the value of variable by different threads.it means multiple threads can use same method and variable at the same time without any problem.

**Lock interface**

It we used as a synchronization mechanism.it is more flexible and provide more method to perform

**SingleTon class**

It’s a class that can only have a one object at a time after the first time , if we try to make another one , it also points to the first instance created.

1. **class** SingletonClassExample

class Singleton {

    private static Singleton instance;

    private Singleton() { }

    public static Singleton getInstance() {

        if (instance == null) {

            instance = new Singleton(); // Lazy initialization

        }

        return instance;

    }

}

**Marker Interface**

Th0ese are the interface that has no methods and constant inside it , it only provide run time information about object.

**Inner Class**

The class is being declared inside a class or iterface

The Cloneable interface in Java is used to indicate that a class allows a clone of its instances to be made. This interface is a marker interface, which means it doesn't contain any methods. It merely signals

Wrapper class:The main purpose of wrapper classes is to provide a way to use primitive data types (like int, char, etc.) as objects.

**Why to write main method static and public?**

We create the main() method as static so that JVM can load the class into the main memory.

We create main() method with **public** access specifier to execute it by any program. So, it is required to define main() method public and if we define the main() method as non-public, it will throw the following error:

how to override and overload main method?

We can overload the main method in java but JVM only calls the original main method, it will never call our overloaded main method.

**No**, we cannot override main method of java because a static method cannot be overridden.

what happens if your serializable class contains a member which is not serializable

It'll throw a NotSerializableException when you try to Serialize it. To avoid that, make that field a transient field. The class will not be serialisable, unless the field is declared transient (which will prevent the field from being serialised, i.e. it will not be saved).

algorithm used for garbage collection in java

The garbage collector uses a mark-and-sweep algorithm to mark all unreachable objects as garbage collection, then scans through live objects to find objects that are still reachable.

During the garbage collection process, the collector scans different parts of the heap, looking for objects that are no longer in use. If an object no longer has any references to it from elsewhere in the application, the collector removes the object, freeing up memory in the heap. This process continues until all unused objects are successfully reclaimed.

Role of finalize method?

"finalize" is a method that is called by the garbage collector before an object is garbage collected and can be overridden to perform any necessary cleanup before the object is destroyed.

Why cant we serialized data into text file?

Whatever serialized data we kept , we kept in as .ser file  and it is in a binary format present in a file.

As it someone open the file .ser as a text file , it will not in readable format , so that’s why if uese as a .ser file then receivver will understand that this is a .ser file we have to deserialiized ith then read it.

What happened when static keyword not included in main method signature

the compilation of the program will go through without any issues but when you'll try to execute it, a "NoSuchMethodError" error will be thrown.

Difference betweeen  int integer , which consmes more memory

The Integer consumes slightly more memory than the 32-bit Java int.

is there multiple finally block in java

There can be only one Finally Block

print something in java without main method

**Using Static Blocks**

**public** **final** **class** **PrintMessageWithoutMainMethod** {

**static** {

System.out.println("Hello World!!"); System.exit(0);

}

}

Reflection in java

Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime.

can a functional interface extend another interface

A functional interface can't extend another interface which has an abstract method, because it will void the fact that a functional interface allows only one abstract method, however functional interface can inherit another interface if it contains only static and default methods in it.

**📌 What Happens If a Functional Interface Breaks the Rule?**

1. **If it has multiple abstract methods**, the compiler **throws an error**:

* java: Unexpected @FunctionalInterface annotation
* The interface will no longer be recognized as a functional interface.

1. **If the @FunctionalInterface annotation is removed**, it becomes a normal interface:

* Can have multiple abstract methods.
* Cannot be used with lambda expressions.

How to prevent singleton from breaking

public class Singleton {   
private static Singleton *instance* = null;   
   
private Singleton() {   
}   
   
public static Singleton getInstance() {   
if (*instance* == null) {   
synchronized (Singleton.class) {   
if (*instance* == null) {   
*instance* = new Singleton();   
}   
}   
}   
return *instance*;   
}   
}

There are 3️⃣ main scenarios that **breaks** ❌ **Singleton** even though we make it **Thread Safe**❗️They are:

1. **Cloning**
2. **Deserialization**
3. **Reflection**

Covarience in java

1. Covariance

Covariance allows a type to be replaced with its subtype. In Java, arrays are covariant.

**Covariance**: Allows a type to be replaced with its subtype. Arrays in Java are covariant.

Jdk ,jre , jvm

**1. JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) that includes two things

* Development Tools(to provide an environment to develop your java programs)
* JRE (to execute your java program).

**2. JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

**3.**[JVM (Java Virtual Machine)](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an [i*nterpreter*](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**.**

Now let us discuss the comp

Jit in java

Just-In-Time (JIT) compilation is an essential part of the Java Virtual Machine (JVM) that significantly improves the performance of Java applications. Here’s a detailed look at JIT compilation in Java:

What is JIT Compilation?

JIT compilation is a process where the JVM compiles Java bytecode into native machine code at runtime. This contrasts with Ahead-Of-Time (AOT) compilation, where the source code is compiled into machine code before execution. JIT compilation enables the JVM to optimize the code for the specific hardware and runtime conditions, leading to improved performance.

can we use private constructor in singleton

Yes

How to make singleton thread safe

We can achieve it by using synchronized block to the core logic that implement new instance

Synchronized(this){

If instancre ==null{

Reurn new}

}

Is clonable is marker interfacce or not , and how?

Here's how Cloneable works as a marker interface:

* When a class implements Cloneable, it indicates that the Object.clone() method is allowed to make a field-for-field copy of instances of that class.
* If a class does not implement Cloneable, calling the clone() method on an instance of that class will throw a CloneNotSupportedException.

Can static block throw exception?

Yes , but if it does then , exception must be handled within a block or a throws clause in it.

Why to use wrapper class :

Arraylist , hashmap others can hold objects , not a premitive data type , wrapper class allow premire data type treated as aobject , enable them to be stored and manged within these collection

Pvsm stringargs

Public makes it acceessible from anywhere , static means no need to create object to call this method , void means it return nothing  , and main is the name of method.

Arinbg args means it is a array that holds any commands line argumenmts to the program

why java is not 100 object oriented programming

No, Java is not a fully object-oriented language as it supports primitive data types like int, byte, long, short, etc., which are not objects. Hence these data types like int, float, double, etc., are not object-oriented. That's why Java is not 100% object-oriented and it has a default value .

How to handle multiple exception

Using pipe operator  while in throws keyword argument we can multiple exception pipe operator

Condition where finally block will not execute :

If jvm exit via system.exit during try catch block

Does finally will run even after return statement in try block

Yes it will

**Instance Variable**

* **Declared inside a class but outside any method.**
* **Scope:** Available throughout the class and accessible by all methods.
* **Lifecycle:** Exists as long as the object exists.
* **Default Values:** Gets default values (e.g., 0 for integers, null for objects).

**Local Variable**

* **Declared inside a method, constructor, or block.**
* **Scope:** Limited to the method or block where it's declared.
* **Lifecycle:** Created when the method starts and destroyed when it ends.
* **Default Values:** Must be initialized before use.

jdk is platform dependent or independent

Answer: JDK and JRE are platform-dependent, while JVM is platform-independent.

need of default and static methods in interfaces

·  **Default Methods:** Allow adding new methods to interfaces without breaking existing implementations, supporting multiple inheritance of behavior.

·  **Static Methods:** Provide a way to include utility methods within interfaces, keeping related methods organized and preventing the need for separate utility classes.

Defualt methods

Reflection in Java

Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime.

Reflection can be used to get information about class, constructors, and methods as depicted below in tabular format as shown:

We use two methods for this purpose as described below before moving ahead as follows:

1. getDeclaredMethod()
2. invoke()

How to make thread sleep for5 min

Thread.sleep(5000);

weak reference vs strong reference java

Its like an way to create new object and how it is behaved ,

And its like its characteristtics are related to garbage collection mechanism.

**Strong References:**This is the default type/class of Reference Object. Any object which has an active strong reference are not eligible for garbage collection. The object is garbage collected only when the variable which was strongly referenced points to null.

Two different levels of weakness can be enlisted: Soft and Phantom

* **Soft References:** In Soft reference, even if the object is free for garbage collection then also its not garbage collected, until JVM is in need of memory badly.The objects gets cleared from the memory when JVM runs out of memory.
* **Phantom References:**The objects which are being referenced by phantom references are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called ‘reference queue’ .

Can we create server in java application without using spring

Yes , only using java se api , by utilizing httpserver class for http services

Can we start thread twiice

No, attempting to start a thread which was already run , will throw illegalthreadstateexceptiuon

Difference between http & httpps

https is just http with encryption and verification.

String Literals

String literals are sequences of characters enclosed between double quote ("") marks. These characters can be alphanumeric, special characters, blank spaces, etc.

Examples: "John", "2468", "\n", etc.

Couopling in java

how often do changes in class A force-related changes in class B.

class loader in java

The Java Class Loader, part of the Java Runtime Environment,

dynamically loads Java classes into the Java Virtual Machine.

Diff between checked and unchecked exception

Checked exceptions in Java must be either caught or declared in the method signature, representing conditions outside the program’s control. Unchecked exceptions, typically caused by programming errors, are not required to be caught or declared.

Stream api , ways to implement lambda expression

1. Stream API is a way to express and process collections of objects.
2. Enable us to perform operations like filtering, mapping, reducing, and sorting.

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);   
   
Stream<Integer> stream1 = numbers.stream(); // create a stream from a collection   
   
Stream<Integer> stream2 = Stream.of(1, 2, 3, 4, 5);

How thread pool works

1. **Creation of Thread Pool**

Java provides the ExecutorService interface to create thread pools, and the Executors utility class offers various factory methods to create different types of thread pools.

java

Copy code

ExecutorService executor = Executors.newFixedThreadPool(5);

In this example, a thread pool with 5 threads is created. This means that at most 5 threads can be active at a time, handling tasks.

2. **Task Submission**

Tasks are submitted to the thread pool using execute() or submit() methods. The execute() method is typically used for tasks that don’t return a result, whereas submit() can return a Future object if the task is expected to return a result.

java

Copy code

executor.execute(() -> {

    // Task code here

    System.out.println("Task is running");

});

or

java

Copy code

Future<Integer> future = executor.submit(() -> {

    return 42;

});

3. **Task Handling**

* When a task is submitted, the thread pool checks if there is any idle thread available in the pool.
* If a thread is available, the task is assigned to that thread.
* If all threads are busy, the task is placed in a queue until a thread becomes available.

4. **Reusing Threads**

Once a thread finishes executing a task, it is not destroyed. Instead, it is returned to the pool and is ready to handle the next task from the queue.

5. **Shutting Down the Thread Pool**

When you're done using the thread pool, it's important to shut it down to free up resources. The shutdown() method will stop accepting new tasks and let the currently executing tasks complete before shutting down.

java

Copy code

executor.shutdown();

If you need to forcefully stop all executing tasks, you can use shutdownNow().

Types of Thread Pools in Java



**Fixed Thread Pool (**newFixedThreadPool**)**: A pool with a fixed number of threads. Tasks are queued if all threads are busy.



**Cached Thread Pool (**newCachedThreadPool**)**: A pool that dynamically adjusts the number of threads depending on the workload. Idle threads are terminated if not used for a while.



**Single Thread Executor (**newSingleThreadExecutor**)**: A thread pool with only one thread. Tasks are executed sequentially.



**Scheduled Thread Pool (**newScheduledThreadPool**)**: A pool that allows tasks to be scheduled to run after a delay or periodically.



Example of Thread Pool in Action

java

Copy code

import java.util.concurrent.ExecutorService;import java.util.concurrent.Executors;

public class ThreadPoolExample {

    public static void main(String[] args) {

        ExecutorService executor = Executors.newFixedThreadPool(3);

        for (int i = 1; i <= 5; i++) {

            final int taskId = i;

            executor.submit(() -> {

                System.out.println("Task " + taskId + " is being executed by " + Thread.currentThread().getName());

            });

        }

        executor.shutdown();

    }

}

use of thread local

**1. Why Use ThreadLocal?**

* Useful for storing **thread-specific data** (e.g., user sessions, transaction contexts).
* Eliminates the need for **synchronization** since each thread has its own instance.

How imtability works in java

An object is considered immutable if its state cannot change after it is constructed.

What happen if we don’t override thtread class rn method

If you don't override the run() method in a Java thread class, the thread will be created and started, but it won't do anything.

How to schedule a task to run after specific interwal ,

 java timer class?

Java has a built-in scheduling framework called the Java Timer API. This Java scheduler API allows users to schedule tasks for future execution, or to run at specific intervals or at fixed times. The Timer API can be used to run background tasks, process user requests, or perform other system-related functions.

[How can we achieve thread safety in Java?https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers](https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers#how-can-we-achieve-thread-safety-in-java)



There are several ways to achieve thread safety in java - synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes. Learn more at [thread safety tutorial](/community/tutorials/thread-safety-in-java).

[What is ThreadLocal?https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers](https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers#what-is-threadlocal)



Java ThreadLocal is used to create thread-local variables. We know that all threads of an Object share it’s variables, so if the variable is not thread safe, we can use synchronization but if we want to avoid synchronization, we can use ThreadLocal variables. Every thread has its own ThreadLocal variable and they can use it gets () and set() methods to get the default value or change it’s value local to Thread. ThreadLocal instances are typically private static fields in classes that wish to associate the state with a thread. Check this post for small example program showing [ThreadLocal Example](/community/tutorials/java-threadlocal-example).

[Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers](https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers#why-wait-notify-and-notifyall-methods-have-to-be-called-from-synchronized-method-or-block)



When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.



[Why Thread sleep() and yield() methods are static?https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers](https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers#why-thread-sleep-and-yield-methods-are-static)



Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

[How does thread communicate with each other?https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers](https://www.digitalocean.com/community/tutorials/java-multithreading-concurrency-interview-questions-answers#how-does-thread-communicate-with-each-other)



When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource. Check this post to learn more about [thread wait, notify and notifyAll](/community/tutorials/java-thread-wait-notify-and-notifyall-example).

How can we make sre main is the last thread to finish in java program

We can use Thread join() method to make sure all the threads created by the program is dead before finishing the main function. Here is an article about [Thread join method](/community/tutorials/java-thread-join-example).

Can we call run method of thread class

Yes, the run() method of the Thread class can be called in Java

suspend thread execution for 5 sec

Thread.sleep(5000);

how super keyword plays the role in polymorphism , specially in context of polymorphism ??

here when we call overridden method , then we gona update it , or modify it , but if we still want to call parent class function , then override it , at that scenario , we can use super keywords.

we can have only one abstract method in functional interface , then what is the need of the functional interface

A functional interface is necessary because it allows you to use lambda expressions, which can only represent a single method implementation, by ensuring that the interface has only one abstract method; this enables cleaner, more concise code for functional programming in Java,

**Strong Reference vs. Weak Reference**

**1. Strong Reference**

A **strong reference** is the default type of reference in most programming languages. When an object is assigned to a variable, it holds a strong reference to that object. The object will not be garbage collected as long as a strong reference to it exists.

**2. Weak Reference**

A **weak reference** does not prevent the object from being garbage collected. This is useful in caching and memory-sensitive applications where we do not want objects to stay in memory unnecessarily.

Static and default method and how to use it ?

public interface Calculator {

    int add(int a, int b); // Abstract method

    default int subtract(int a, int b) { // Default method

        return a - b;

    }

    static double PI = 3.14159; // Static variable

    static double calculateCircleArea(double radius) { // Static method

        return PI \* radius \* radius;

    }

}

class BasicCalculator implements Calculator {

    @Override

    public int add(int a, int b) {

        return a + b;

    }

    // Can override subtract if needed

}

public class Main {

    public static void main(String[] args) {

        BasicCalculator calc = new BasicCalculator();

        System.out.println(calc.add(5, 3)); // Calls the overridden add method

        System.out.println(calc.subtract(10, 2)); // Uses the default subtract implementation

        System.out.println(Calculator.calculateCircleArea(5)); // Calls the static method

    }

}

when to use e.printStackTrace?

The printStackTrace() method in Java is a tool used to handle exceptions and errors. It is a method of Java's throwable class which prints the throwable along with other details like the line number and class name where the exception occurred. The printStackTrace() is very useful in diagnosing exceptions.

when to use static and default method of interface?

interface MathUtil {

    static int add(int a, int b) {

        return a + b;

    }

}

public class Test {

    public static void main(String[] args) {

        int result = MathUtil.add(5, 3); // Calling without an object

        System.out.println(result); // Output: 8

    }

}

**Example: Default Method in an Interface**

java

CopyEdit

interface Vehicle {

    void speedUp(); // Abstract method

    default void show() {

        System.out.println("This is a vehicle.");

    }

}

class Car implements Vehicle {

    public void speedUp() {

        System.out.println("Car is speeding up.");

    }

}

public class Test {

    public static void main(String[] args) {

        Car myCar = new Car();

        myCar.speedUp();

        myCar.show(); // Uses default method

    }

}

**Use Case**

Utility/helper functions

Backward compatibility with default behavior

composition , association and assertion along with the code and defination in java

**Association**

**Definition:**

Association represents a **relationship between two independent classes** where one class uses another.

* There is **no ownership** (both classes can exist independently).
* It can be **one-to-one, one-to-many, many-to-one, or many-to-many**.

import java.util.\*;

class Student {

    String name;

    public Student(String name) {

        this.name = name;

    }

}

class Teacher {

    String name;

    List<Student> students; // A teacher can have multiple students

    public Teacher(String name) {

        this.name = name;

        this.students = new ArrayList<>();

    }

    public void addStudent(Student student) {

        students.add(student);

    }

    public void showStudents() {

        System.out.println(name + " teaches:");

        for (Student s : students) {

            System.out.println(" - " + s.name);

        }

    }

}

public class AssociationExample {

    public static void main(String[] args) {

        Teacher teacher = new Teacher("Mr. Sharma");

        Student s1 = new Student("Rahul");

        Student s2 = new Student("Priya");

        teacher.addStudent(s1);

        teacher.addStudent(s2);

        teacher.showStudents();

    }

}

Like what we can implement one to many , one to one , .

**Aggregation (Has-A Relationship)**

**Definition:**

Aggregation is a **weaker form of association** where:

* **One class owns another** but the owned object **can exist independently**.
* There is **a “has-a” relationship** (but objects have independent lifetimes).
* If the **container object is destroyed, the contained object still exists**.

import java.util.\*;

class Professor {

    String name;

    public Professor(String name) {

        this.name = name;

    }

}

class Department {

    String deptName;

    List<Professor> professors;

    public Department(String deptName) {

        this.deptName = deptName;

        this.professors = new ArrayList<>();

    }

    public void addProfessor(Professor professor) {

        professors.add(professor);

    }

    public void showProfessors() {

        System.out.println(deptName + " Department has:");

        for (Professor p : professors) {

            System.out.println(" - " + p.name);

        }

    }

}

public class AggregationExample {

    public static void main(String[] args) {

        Department csDept = new Department("Computer Science");

        Professor prof1 = new Professor("Dr. Sharma");

        Professor prof2 = new Professor("Dr. Rao");

        csDept.addProfessor(prof1);

        csDept.addProfessor(prof2);

        csDept.showProfessors();

    }

}

**Key Point:** If the Department is deleted, the Professor objects still exist.

**Composition (Strong Has-A Relationship)**

**Definition:**

Composition is a **stronger form of aggregation** where:

* **One class contains another class** as a part of itself.
* **The contained object cannot exist independently** (dependent on the container).
* **If the container object is destroyed, the contained object is also destroyed.**

class Room {

    String roomType;

    public Room(String roomType) {

        this.roomType = roomType;

    }

    public void showRoom() {

        System.out.println("Room Type: " + roomType);

    }

}

class House {

    private Room room1;

    private Room room2;

    public House() {

        room1 = new Room("Bedroom");

        room2 = new Room("Kitchen");

    }

    public void showHouse() {

        room1.showRoom();

        room2.showRoom();

    }

}

public class CompositionExample {

    public static void main(String[] args) {

        House house = new House();

        house.showHouse();

        // If house is destroyed, rooms are destroyed too

    }

}

**Key Point:** Room objects cannot exist without House.

how hashset insecure no duplicates

HashSet in Java ensures no duplicates by using the equals() and hashCode() methods to compare objects and determine if they are considered equivalent, effectively preventing the storage of duplicate elements.

* **How it Works:**
* When you add an element to a HashSet, the add() method first calculates the element's hash code using the hashCode() method.
* Then, it uses this hash code to determine the index (or bucket) where the element should be stored.
* Before storing the element, HashSet checks if an element with the same hash code already exists in that bucket.
* If an element with the same hash code exists, the equals() method is called to compare the two objects.
* If equals() returns true, the HashSet considers the objects to be equal, and the new element is not added (duplicate avoided).
* If equals() returns false, the objects are considered different, and the new element is added to the HashSet.

Weak hashmap

WeakHashMap is an implementation of the Map interface. WeakHashMap is almost the same as HashMap except in the case of WeakHashMap if the object is specified as the key doesn’t contain any references- it is eligible for garbage collection even though it is associated with WeakHashMap. i.e Garbage Collector dominates over WeakHashMap.

An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use.

When to use :

**For Storing Metadata**: Used in frameworks to associate temporary metadata with objects.

use of static keyword?:

The static keyword in Java is used for memory management and to create class-level members. It can be applied to variables, methods, blocks, and nested classes.

Ways to encapsulate data in java:

Or bean class of java :

Its like an way to write a class code in a different manner like

Make field private and final

Allow exposure of class only through the public method

Initialize value only through constructor .

Inheritance vs composition:

Inheritance : it is a “IS A” relationship dog is a animal

Composition: it is a “HAS A” relationship car has a engine

Predefine interface

Predicate : it has one Boolean test method , it takes input parameter and return Boolean value.

Function: it takes two input and return output after functioning on it. It has method apply

Consume: it is to take input and function on it but did not return anything in return .it has accept method.

Supplier: it has get method . it did not take any input but return in something.

BiPredicate: its like same predicate , but it takes two parameters. It has method named test.

Bifunction : same like function , but it takes two inputs , and this also has apply method .

Biconsumer: same like above.

SOLID

S-> single responsibility principle.means one class should be stick to only one kind of functionality.

o->open closed principle. Open for extension but closed for modification.

L-> liskov substitive principle : it means subtype should be replaced by base type.without breaking of behaviour of subtype.

I->Interface segregation principle. It means don’t forced the class to implement methods that doesn’t need.

D-> dependency inversion principle.means our main logic should not be forcefully coupled to low level code . it should maintain class-interface structure and at use , use interface instance.