1)Difference between JVM, JRE and JDK?

JVM is the engine that runs java bytecode and making it platform-independent.

JRE contains the JVM and the standard libraries that java programs needs to run.

JDK is development kit for developers that contains everything like JRE and required tools like compilers, debuggers to create java applications.

2)Key components of JVM architecture?

JVM has three components, the class loader, the runtime data areas and the execution engine.

The class loader loads class files into the JVM.

The runtime data areas stores the data needed while the program runs, like memory for variables and code.

The execution engine runs instructions in the class files.

Memory storages available with JVM:

Heap space, Stack memory, Method area and Native method stacks.

Heap space in java is where the program stores objects and data that it creates and shares.

Stack memory is used for keeping track of what happens inside each function call. Including variable values.

Method area stores the information about the program’s classes like methods and constants.

Iq) What is meta space in java?

Meta space is the area of memory where the JVM stores class metadata, which includes information about class names, methods, fields, and other class level data.

Before java 8, class metadata was stored in a fixed-size memory area called PermGen(Permanent Generation).Metaspace was introduced in java8 to address the limitations and problems associated with PermGen, such as the need to manually configure its size and the risk of ‘out of memoryError’ due to insufficient space . This enhancement helps improve the performance and stability of java applications.

3)How does garbage collector works in java?

Garbage collection in java automatically frees memory by removing unused objects.

The finalize() is called by garbage collector on an object. When it determines that there are no more references to the object.

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

4)Explain Public static void main(String args[])?

Public makes this method accessible from anywhere, static means I don’t need to create an object to call this method, void means it doesn’t return any value, and main is the name of this method.

What happens if we don’t declare the main as static?

If we don’t declare the main method as static , the jvm won’t be able to launch the application.

As a result the program will compile, but it will fail to run, giving an error like “main method is not static in class class name”

5)can we override or overload main method?

We cannot override main method of java because a static method cannot be overridden.

Static methods in java associated with class where as the non static method associated with object. Static belongs to the class area, static method don’t need an object to be called.

We can overload main method in java by just changing the arguments.

6)Difference between primitive and non-primitive data type in java?

Primitive data types in java are the basic types of data predefined by the language and named by a keyword. They have fixed size and are not objects.

Example: byte, short, int, long, double, char and Boolean.

Note: primitive data types cannot be null, they have default values (e.g 0 for int, false for Boolean, 0.0 for double)

Non-primitive data types are objects and classes that are not defined by java itself but rather by the programmer or the java Api. They can be used to call methods to perform certain operations, and their size is not fixed.

Example: String, array and any other class instances.

7)why we use wrapper classes in java?

Because java collections, such as Arraylist, HashMap and others in java collection framework can only hold objects.

8)What is unboxing and autoboxing?

Auto-boxing automatically converts a primitive type(like int) to its corresponding wrapper class(Integer). Unboxing does the reverse, converting an Integer to int.

9) role of each try, catch and finally block in exception?

Try block contains the code that might throw exceptions, catch handles those exceptions, finally executes code after try/catch, regardless of an exception, typically for clean up.

10) can we have multiple finally blocks in java?

No, each try can have only one finally block, multiple finally blocks are not allowed within a single try-catch finally structure.

11)checked and unchecked exceptions?

Exception is an unwanted event that occurs during the execution of program and disrupts the flow.

Checked exceptions must be declared or handled.

Unchecked exceptions do not need to be declared or caught.

12)How to handle multiple exceptions in a single catch block?

We can use a single catch block for multiple exceptions by separating them with a pipe(|), e.g catch(IOException | SQl Exception e), to handle both exceptions with a same logic.

13)String pool in java?

A java string pool is a place in heap memory where all strings defined in the program are stored. Whenever we create a new string object, JVM checks for the presence of the object in the string pool. If string is available in the pool the same object reference is shared with the variable else a new object is created.

14)Access modifiers?

To control access to classes, methods and fields, ensuring appropriate visibility and encapsulation.

Public : accessible from anywhere

Protected: accessible to sub classes and classes within the package.

Default: only within the package.

Private: within their own class.

15)Classes and objects in java?

Classes are blue prints for objects in java, defining the state and behavior that the objects of the class can have. Objects are instances of classes, representing entities with states and behavior defined by their class.

Ways to create objects:

Using new key word

Using class factory methods

Using the clone()

16)Single ton class?

A single ton class in java is a class that can have only one instance (or object) at any time. It is useful when we want to make sure there’s just one shared resource, like a configuration setting or a connection to a database.

In order to create a singleton class, first we need to make a constructor as private, next we have to create a static instance of the class and finally we have to provide static method instance so that’s how we can create the singleton class.

Are this thread safe?

Single ton classes are not thread safe by default. If multiple threads try to create an instance at the same time, it could result in multiple instances. To prevent this, we can synchronize the method that creates the instance or use a static initializer.

17)What is constructor?

A constructor in java is a special method used to initialize new objects. It has same name as class and may take arguments to set initial values for the object’s attributes.

Can we use a private constructor?

We can use private constructor in java. They are mostly used in classes that provide static methods or contains only static fields. A common use is in the single ton design pattern, where the goal is to limit the class to only one object.

Can constructor be overloaded?

We can have multiple constructors in java class, each with a different set of parameters. This lets us creates objects in various ways depending on what information we have at the time.

18)What is immutability

Immutability in java means that once object’s state is created, it cannot be changed.

Why immutable objects are useful for concurrent programming?

These are useful in concurrent programming because they can be shared between threads without needing synchronization.

What are immutable classes in java?

Immutable classes in java are classes whose objects cannot be modified once it is created. This means all their fields are final and set only once, typically through the constructor.

How can we create a immutable class?

1. Declare the class as final so it cannot be extended.
2. Make all the fields final and private so that direct access is not allowed.
3. Don’t provide any setter methods for variables.
4. Initialize all fields using a constructor method.

19) What is inheritance in java?

Inheritance in java means a class can use the features of another class. This helps to reuse code and make things simpler.  
Java doesn’t support multiple inheritance to avoid ambiguity.

Difference between inheritance and composition?

Inheritance in when one class gets its features from another class.

Composition is when a class is made using parts from other classes which can be more flexible.

20)what is polymorphism in java?

Polymorphism in java means that the same piece of code can do different things depending on what kind of object it is dealing with. For example, if you have a method called “draw” it might make a circle for a circle object and a square for a square object.

How does method overloading relate to polymorphism?

Method overloading is using the same method name with different inputs in the same class.

What is dynamic method dispatch in java?

Dynamic method dispatch is a way to java decides which method to use at run time when methods are overridden in sub classes. It ensures the correct method is used based on the type of object.

Can constructors be polymorphic?

No constructors cannot be polymorphic. We can have many constructors in a class with different inputs, but they don’t behave differently based on the object type like methods do.

Is it possible to overload methods that differ only by their return type in java?

We cannot overload methods by changing their return type. The methods must differ by their parameters for overloading to be valid.

What is method overriding in java?

To override a method, the new method in the sub class must have the same name, return type, and parameters as the method in the parent class. Also the new method should not be less accessible than the original.

When we call an overloaded method, the java compiler looks at the number and type of arguments we have provided and picks the method that matches these arguments best.

21) what is abstraction?

Abstraction in java means focusing on what needs to be done, not how to do it. You create a kind of blueprint that tells other parts of program what actions they can perform without explaining the details.

Example of abstraction:

Java uses abstraction in its collection tools. For example, when you use a list, we don’t need to know how it stores data, whether as an array list or a linked list.

What happens if a class includes an abstract method?

A class with an abstract method must itself be abstract. We can’t create objects directly from an abstract class, it is meant to be a blueprint for other classes.

22) What is inheritance in java?

Interface is like a blueprint for a class. It defines a set of methods that the class must implement without specifying how these methods should work.

Difference between an interface and abstract class?

Abstract class achieves partial abstraction(0 t0 100%) where as interface achieves fully abstraction. Abstract classes can have abstract and non- abstract methods where as interfaces can have only abstract methods(Since java 8, it can have default and static methods also).

Example for when to use interface and when to extend a class?

Use an interface when we want to list the methods a class should have, without detailing how they work. Use class extension when we want a new class to inherit features and behaviors from an existing class and possibly modify them.

23)what is encapsulation in java?

Encapsulation in java is like putting important information into a safe. We store data and the methods inside a class, and we control who can access or change the data by using specific methods.

How encapsulation enhances software security and integrity?

Encapsulation keeps important data hidden and safe. It only lets certain parts of our program use this data, which helps prevent mistakes and keeps the data secure from unwanted changes.

24) What is ‘this’ and ‘super’ keyword in java?

‘this’ is used to refer current class instance as well as static members.

‘super’ is used to access methods of the parent class.

What happens if we attempt to use the ‘super’ keyword in a class that doesn’t have a super class?

A compilation error occurs. The “super” keyword is only applicable within subclasses to refer to members of the super class.

Can we use this or super keyword in static methods?

No, this and super keywords cannot be used in static methods. Static methods belongs to the class, not instances, and super refers to the super class’s object context, which does not exist in a static context.

25)Static keyword in java?

Static key word in java is used to indicate that a particular member (variable or method) belongs to the class, rather than any instance of the class. This means that the static member can be accessed without creating an instance of the class.

Can a static block throw an exception?

Yes, It can throw but if it does, the exception must be handled within the block itself or declared using a throws clause in the class.

Can we override static methods?

No, we cannot override static methods, because method overriding is based on dynamic binding at runtime and static methods are bound at compile time.

Is it possible to access non-static members from within a static method?

Yes, it is possible to access non-static members from within a static method by creating an instance of the class containing those members.

What is static block?

To initialize static variables, the statements inside static block are executed only once, when the class is loaded in the memory.

Why do we need static methods inside interfaces?

Static methods in interfaces can be used to provide utility and helper methods related to the functionality of the interface. These methods can offer common functionality that can be reused by any class implementing these interfaces or even by other classes without requiring an instance of the interface.

26)What is final keyword?

The final key word is used to declare constants, making variables unchangeable once assigned, or to prevent method overriding or class inheritance.

27)What is functional interface?

Functional interfaces are interfaces in java that they have only one abstract method. They are used to represent behaviors as objects, enabling functional programming in java. They can contain default or static methods but must have only one abstract method to qualify as functional.

Can functional interfaces extend another interface?

No, functional interfaces allow to have only single abstract method. However functional interface can inherit another interface if it contains only static and default methods in it.

28)Some new features introduced in java 8?

Lambda expressions, stream API, method references, Default methods, Optional class, New date and time API are the new features introduced in java 8.

Optional class: introduced as a way to address the problem of null references.

Lambda Expression: Introduced to make it easier to write code for interfaces that have only one method (Functional interface)

Stream API: Introduced to help developers to process collections of data in a more efficient way, especially bulk operations like filtering or sorting.

29) Difference between filter and map functions in Stream API?

Filter() eliminates elements of collection where the condition is not satisfied where as Map() method is used to perform operation on all elements hence, it returns all elements of collection.

30) new features introduced in java 11, java 17 and java 21?

31) What is collection framework?

If we want to collect group of objects as a single entity then we should go for collections.

The main parts of java collection framework are interfaces like collection, List, Set, Queue and Map. Each one helps manage data in different ways.

How Iterator works within the java collection framework?

An iterator is a tool in the collection framework that lets us go through a collection’s elements one by one.

Some common methods in all collection types?

Some common methods all collection types have are add, remove, clear, size and isEmpty. These methods let us add and remove items, check the size, and see if the collection is empty.

How does collection framework handle concurrency?

The collection framework deals with multiple threads using special collection classes like concurrentHashMap and copyOnWriteArraylist, which set different parts of our program modify the collection at the same time safely.

32)How can we choose the right collection type for a specific problem?

To pick the right collection type, think about what we need:

List : if we want an ordered collection that can include duplicates.

Set: If we need unique elements that cannot have duplicates.

Queue: For processing elements in order, and

Map: for storing pairs of key and values.

What enhancement were made to the java collection framework in java 8?

Adding streams, which makes it easier to handle collections in bulk, and lambda expression, which simplify writing code for operations on collections.

Difference between iterator and list iterator?

Iterator allows forward traversal of a collection, while Listiterator extends iterator functionality to allow bidirectional traversal of lists and also supports elements modification.

Name the algorithms used by Arrays.sort() and Collection.sort()?

Arrays. Sort() uses Dual-pivot Quicksort algorithm for primitive types and TimSort for objects arrays.

Collection.sort() uses TimSort, a hybrid sorting algorithm combining merge sort and insertion sort.

33)Internal working of Hash map?

**Map<Employee, String> map= new HashMap<>();**

When we create a hash map object, internally it will create a 16 buckets structure (since initial capacity of map is 16 (0-15)).

Each bucket internally uses a linked list (Each bucket can be considered as a linked list)

Each linked list can have n number of nodes and the node structure will be key, value, hash, next.

Now we will add entry to hash map object:

map.put(emp1,”Dev”)

when we try to add an entry to map object, the hashFunction takes hash code of a key and generates a number. This number tells in which bucket the entry will be stored.

Note: How hashing works internally:

Whenever we try to store entry in hash map object it will take the hash code of the key object and apply modulo operation with the number of buckets. For example, if there are 16 buckets, we take hash code and calculate ‘**hashcode % 16’** .

It will generate a number by performing above operation and it stores the entry in that bucket.

What happens If two keys have the same hash code?

Can we include class as a key in hashMap?

Yes, we can use a class as a key in Hashmap. However, when using a class as a key, it’s important to ensure that the class overrides the hashcode() and equals() methods properly to ensure correct functioning of the hashmap.

33) What is design pattern in java and why we use this?

Design patterns are proven solutions for common software design pattern problems.

They provide standardized approaches to organize code in a way that is maintainable, scalable and understandable.

Few common design patterns used in java programming ?

Singleton: Ensures a class has only one instance, with a global access point.

Observer: Allows objects to notify others about changes in their state.

Factory methods: Delegates the creation of objects to sub classes, promoting flexibility.

34)What is a transaction in DB?

A database transaction is a sequence of operations performed as a single logical unit of work. A transaction must be either fully completed or fully failed; it cannot be left in an intermediate state. Transactions are used to ensure data integrity and consistency.

ACID properties?

1.Atomicity: Ensures that all operations within a transaction are completed successfully. If any operation fails, the entire transaction is rolled back, and no changes are applied to the database.

Ex: Imagine if we re transferring 100 from savings account to checking account. Atomicity ensures that either the 100 is fully taken from our savings account and added to our checking account, or if something goes wrong , nothing changes in either account.

2.Consistency: Ensures that transaction transforms the database from one valid state to another. The integrity constraints of the database must be maintained before and after the transaction.

Ex: After the transfer of 100, our total balance across both accounts should be correct and follow the bank’s rules.

3.Isolation: Ensures that operations of a transaction are isolated from the operations of other transactions. Intermediate states of a transaction are not visible to other transactions. This prevents issues like dirty reads, non-repeatable reads, and phantom reads.

Ex: If two people transferring money from the same account at the same time, isolation ensures that your transfer don’t get mixed up. Each of our transactions looks like it happened one after the another, even though they were processed simultaneously.

Durability: Ensures that once a transaction is completed the changes are permanent, even in the event of system failure. The changes made by a committed transaction are saved and persisted in the database.

EX: once the transaction is confirmed, it should stay recorded, even if the bank’s computer system crashes right after.

What are SOLID Principles?

“S” stands for single responsibility principle: It means a class should only have one reason to change, meaning it should handle just one part of the functionality.

For example: A class vehicle registration should only handle vehicle registration details. If it also takes care of vehicle insurance, then it will violate this.

“O” stands for open/closed principle: It means classes should be open for extension but closed for modification.

“L” stands for Liksov substitution principle: It means objects of a super class should be replaceable with objects of its sub class without affecting the program’s correctness.

“I” Interface segregation principle: It means do not force any client to depend on methods it does not use; split large interfaces into smaller ones.

For example: instead of one large interface vehicleOperations with methods like drive, refuel,charge and navigate etc…;

Split it into focused interfaces like drivable, refuelable and navigable etc..,

An electric car wouldn’t need to implement Refuelable, just chargeable and Navigable.

“D” stands for Dependency inversion principle: It means high level modules should not depend directly on low-level module but communicate through abstraction like interfaces.

Ex: Suppose we have a ‘LightBulb’ class and a ‘switch’ class that directly depends on it. If we want to change ‘lightBulb’ to ‘LED’, we must change ‘switch’ too. Instead, we can use an abstraction.

interface Switchable {

void turnOn();

void turnOff();

}

class LightBulb implements Switchable {

public void turnOn() {

// LightBulb turns on

}

public void turnOff() {

// LightBulb turns off

}

}

class LED implements Switchable {

public void turnOn() {

// LED turns on

}

public void turnOff() {

// LED turns off

}

}

class Switch {

private Switchable device;

public Switch(Switchable device) {

this.device = device;

}

public void operate() {

// Switch operates the device

device.turnOn();

}

}

**Exceptions in java:**

Exception in java is an abnormal condition that occurs during the execution of program and disrupts the normal flow of the program. If not handled properly it can cause the program to terminate abruptly.

**How do we handle exception?**

We can handle exceptions using 3 blocks:

Try: Encloses set of statements which can throw exception hence are required to be monitored.

Catch: This block catches exception when it occurs and work accordingly to handle it or throw it as required.

Finally: This block gets executed always regardless of exception occurrence. Hence clean up is done here.

**Hierarchy of exception?**

**Throwable**

ErrorException

->stack overflow error checked exception unchecked/Runtime exception

->out of memory error Io Exception Null pointer exception

->virtual memory error SQL Exception Number format exception

Class not found Exception Index out of bounds exception

File not found Exception Arithmetic exception.

**Checked Exception:**

These exceptions are checked at runtime and handled at the time of compilation.

These exceptions are direct sub classes of the exception class.

Checked exception require handling using a try-catch block or at least the method should use the throws keyword to let the calling method know that a checked exception could be thrown from this method.

unchecked exception:

These are not checked and handled at compile time. They occur during the runtime of the program.

These exceptions are a direct subclass of the Runtime Exception class.

This exception could occur due to mistakes in programming logic.

**Can we write only try block without catch and finally blocks?**

No, either catch or finally is must. If we use try without catch or finally block, we will get compile time error saying “insert finally to complete try statement” like this

**Can we write any other statements between try catch or finally block?**

No. Try must be followed directly by either catch or finally. If we add some statements in between try and catch or finally block, we will get co0mpile time error saying, “insert finally or catch block to complete try statement”.

**Does remaining statements in try block executes after exception occurs?**

No, if exception occurs at a particular point in try block, then all statements after that statement where exception occurs will not be execute and the flow goes directly to either catch block if there is any or else program terminates. Hence we need finally block to do all clean up like closing files or removing locks.

**Diff between throw and throws**.

Throw:

Java throw key word is used to explicitly throw an exception.

Checked exception cannot be propagated using throw only.

Throw is used within the method.

We cannot throw multiple exceptions.

Throws:

Java throws keyword to declare an exception.

Checked exception can be propagated with throws.

Throws is used with method signature.

We can declare multiple exceptions.

**What happens when an exception is thrown by the Main method?**

When an exception thrown by the main() method, the java Runtime terminates the program and prints the exception message and the stack trace in system console.

**What do you understand by unreachable catch block error?**



This error comes when we keep super classes first and sub classes later. Like here we kept Exception first which is parent of Null pointer Exception.

Hence the order of catch blocks must be from most specific to most general ones.

What is multi catch block?

To reduce code duplication and makes it easier to maintain, java 7 came up with multi catch block concept.

We can have multiple exceptions piped in a catch block as arguments.

Ex:

Try {

//statements

}

Catch (NullPointerException | SQLException ex){

//----------

}

Diff between Final, Finally and Finalize?

Final: It is a key word used apply restrictions on the class, method, variable. The final class can’t be inherited final method can’t be overridden and final variable can’t be changed.

Finally: this key word is used with the try-catch block to provide statements that will always get executed even if some exception arises. Usually, finally is used to close resources.

Finalize: is used to perform clean up processing just before the object is garbage collected.

What is exception propagation?

Now consider a situation where database exception occurs, and u need to tell this to user. You can see multiple layers in between. How to preserve the original exception when it propagates across multiple logical layers in a program.

The answer is through exception chaining.

Here database throws SQL Exception. Which is caught by DAO layer.

DAO class throws DAO Exception which is caught by catch block in service class.

Service class throws the catched DAO Exception as Service Exception which in turn is caught by controller.

Controller will now handle it appropriately and send the user friendly message to UI to be shown.

The printed exception stack trace reveals an exception propagates from the SQL/DB layer up to the controller layer.

This is called exception propagation which helps in debugging process.



**What is exception chaining?**

Exception chaining happens when one exception is thrown due to another exception. This helps developers to understand under what situations an exception was thrown that in turn caused another exception in the program.

EX: Consider online shopping example

1. User places an order.
2. System checks the inventory.
3. System processes the payment.
4. System confirms the order.

Step 1: Placing order.

Once the user places an order.

No exception occurs here.

Step 2: check inventory.

The system checks the inventory and finds the items are available.

No exception occurs here.

Step 3: process payment

The system tries to process the payment. The payment fails because the user’s account has insufficient funds.

This throws ‘In sufficient exception’.

Ste4: Handle payment failure

The system catches ‘Insufficient funds exception’.

To give more context about where the failure happened, it wraps this exception in a new ‘order processing Exception’ and includes a message saying, “payment failed during order processing”.

Step4: confirms the order.

The system tries to confirm the order. Since the payment failed, confirming the order also fails.

The system catches the “order processing exeption”.

**Note:**

->Original Exception: “Insufficient funds exception” is thrown when the payment fails.

->Chained Exception: “OrderprocessingException” is thrown with “InsufficientFundsException” as the cause, providing more context.

This way, exception helps keep track of what went wrong and where, making it easier to debug and understand the issue.

**How to create custom Exception class?**

By extending the ‘**Exception’** or ‘**RunTimeException’** (which itself extends Exception).

**Step1**:- Decide whether our custom exception should be checked(‘Exception’) or unchecked(‘RunTimeException’).

**Step2:** create a new class that extends ‘Exception’ or ‘RunTimeException’. We have to name it in a way that reflect the specific type of exception that we want to represent.

**Step3**:- Provide constructor that call the super class constructors(**super())** to initialize the exception message and possibly other exceptions.

**Note:** why we have a choice to extend to extend either **‘Exception’(For checked Exception) and ‘RunTimeException’** for **(Unchecked exception).**

**Why might we choose to extend Runtime exception (even if runtime exception extending Exception)?**

By extending RunTimeException our custom exception becomes an unchecked exception. This means that is does not need to be explicitly caught or declared in method signatures, offering more flexibility and convenience in programming.

Runtime exceptions and its sub classes indicates programming errors, such as invalid arguments(‘Illegal argument Exception’) or accessing null reference (‘Null Pointer Exception’), which are typically bugs that application should avoid.

While we technically can extend Exception for both checked and unchecked exceptions, It is a convention in java to extend ‘RunTimeException’ for un checked exceptions.

**Multi-threading:**

**What is process and thread?**

Thread is a smallest unit of execution within a process or program.

Many threads can run concurrently within a program.

At run time, threads in a program exist in a common memory space and can, therefore, share both data and code(i.e., they are light weight compared to processes).

Note: The programs which we were running so far, all those programs are single threaded. Why because java has its thread called main thread.

Process is an independent program in execution.

**Multi tasking:**

Multi tasking allows several activities to occur concurrently on the computer.

1. Process-based multi tasking(multiple programs or processes running parallelly)
2. Thread-based multi-tasking(multiple threads run within a program)

Process-based multi-tasking allows processes (i.e programs) to run concurrently on the computer.

Eg: Running the Ms paint while also working with the word processor.

Thread-based multi-tasking allows parts of the same program to run concurrently on the computer.

Eg: MS word that is printing and formatting text at the same time and suggesting spellings of the word is not correct.

**Thread vs Process:**

Two threads share the same address space.

Context switching between threads is usually less expensive than between processes.

The cost of communication between threads is relatively low.

**Why multi-threading?**

In a single threaded environment, only one task at a time can be performed. So, CPU cycles are wasted, for example, when waiting for user input.

Multi-tasking allows idle CPU time to be put to good use.

**THE MAIN THREAD:**

When a standalone application runs, a user thread is automatically created to execute the main() method of the application. This thread is called main thread.

If no other user threads are spawned, the program terminates when the main() method finishes executing.

All other threads, called child threads, are spawned from the main thread.

The main() method can then finish, but the program will keep running until all user threads have completed.

The runtime environment distinguishes between user threads and daemon threads.

**Note:**

If there is no user threads running program will stop irrespective of whether the daemon threads are running or not.

Calling the setDaemon(Boolean) method in the thread class marks the status of the thread as either daemon or user, but this must be done before the thread is started.

As long as user threads is alive, the JVM does not terminate.

Daemon threads are referred to as a low priority threads,these are required for supporting background tasks like garbage collection, releasing memory of unused objects etc.,

**How to create our own thread:**

A thread in java is represented by an object of the thread class.

Creating threads is achieved in one of two ways.

1. Implementing the java.lang.Runnable interface.

->create a userdefined class and make that class extends runnable interface.

->override run method.

->instantiate an object (create object) of a class that implements the runnable interface.

->create a Thread object and pass our runnable object to its constructor.

public class MyRunnable implements Runnable {

@Override

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(Thread.currentThread().getId() + " Value " + i);

}

}

public static void main(String[] args) {

MyRunnable myRunnable = new MyRunnable();

Thread thread1 = new Thread(myRunnable);

Thread thread2 = new Thread(myRunnable);

thread1.start();

thread2.start();

}

1. Extending the java.lang.Thread class.

->Create an userdefined class and extends with thread class

->override run() method

->create instance of userdefined class.

->start the thread.

public class MyThread extends Thread {

@Override

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(Thread.currentThread().getId() + " Value " + i);

}

}

public static void main(String[] args) {

MyThread thread1 = new MyThread();

MyThread thread2 = new MyThread();

thread1.start();

thread2.start();

}}

**Synchronization in java:**

Threads share same memory space, i.e. they can share resources(Objects).

Note: Threads can share resources, so it allows multiple threads to access same resource.

However, there are some critical situations where it is desirable that only one thread at a time has access to a shared resource. So here synchronization allows only one thread to access the resorce.

While a thread is inside a synchronized method of an object, all other threads that wish to execute this method will have to wait.

**The Volatile Keyword:**

In java, the ‘volatile’ keyword is used to indicate that a variables value may be changed by multiple threads simultaneously. The primary purpose of ‘volatile’ keyword is to prevent threads from caching the variables value locally.

When a variable declared as volatile, All reads and writes of the volatile are directly done from and to main memory.

**Producer consumer problem:**

Producer-consumer problem also known as bounded-buffer problem is a classic example of a multiprocess synchronization problem**.**

The problem describes two processes, the producer and the consumer, which share a common, fixed-size buffer used as a queue

* The producers job is to generate data, put it into the buffer, and start again.
* At the same time, the consumer is removing data from the buffer.

**Problem:**

To make sure that the producer wont try to add data into the buffer if its full and that the consumer wont try to remove data from an empty buffer.

Solution:

The producer is to either go to sleep if the buffer is full. The next time the consumer removes an item from the buffer,It notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer.

**public** **class** Threadexample {

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

**throws** InterruptedException

    {

        // Object of a class that has both produce()

        // and consume() methods

**final** PC pc = **new** PC();

        // Create producer thread

        Thread t1 = **new** Thread(**new** Runnable() {

            @Override

**public** **void** run()

            {

**try** {

                    pc.produce();

                }

**catch** (InterruptedException e) {

                    e.printStackTrace();

                }}

        });

        // Create consumer thread

        Thread t2 = **new** Thread(**new** Runnable() {

            @Override

**public** **void** run()

            {

**try** {

                    pc.consume();

                }

**catch** (InterruptedException e) {

                    e.printStackTrace();

                }}

        });

   t1.start();

        t2.start();

        // t1 finishes before t2

        t1.join();

        t2.join();}

**public** **static** **class** PC {

        // Create a list shared by producer and consumer

        // Size of list is 2.

        LinkedList<Integer> list = **new** LinkedList<>();

**int** capacity = 2;

        // Function called by producer thread

**public** **void** produce() **throws** InterruptedException

        {

**int** value = 0;

**while** (**true**) {

**synchronized** (**this**)

                {

                    // producer thread waits while list

                    // is full

**while** (list.size() == capacity)

                        wait();

                    System.out.println("Producer produced-"

                                       + value);

                    // to insert the jobs in the list

                    list.add(value++);

                    // notifies the consumer thread that

                    // now it can start consuming

                    notify();

                    // makes the working of program easier

                    // to  understand

                    Thread.sleep(1000);

                }}}

        // Function called by consumer thread

**public** **void** consume() **throws** InterruptedException

        {

**while** (**true**) {

**synchronized** (**this**)

                {

                    // consumer thread waits while list

                    // is empty

**while** (list.size() == 0)

                        wait();

                    // to retrieve the first job in the list

**int** val = list.removeFirst();

                    System.out.println("Consumer consumed-"

                                     + val);

                    // Wake up producer thread

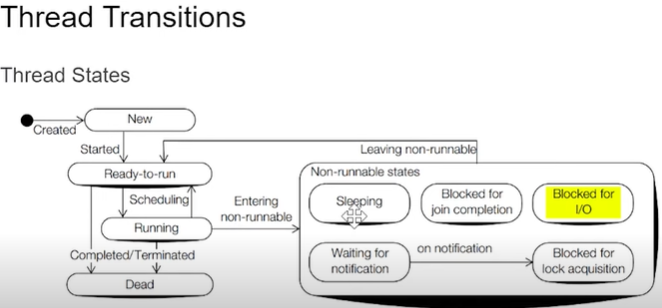
                    notify();

                    // and sleep

                    Thread.sleep(1000);

                }}}}}

**Lifecycle of a thread:**



Note: whenever a thread is notified or awakened it does not directly jump into the runnable state , it means it will go to “blocked for lock acquisition state” and fight for lock then it can start execution.

A close-up of a text

Description automatically generated

Wait(): this method causes the current thread to wait and go to sleep until some other threads call the notify() or notifyll() method for the objects monitor(lock). It simply releases the lock and mostly used for inter-thread communication. It should only be called from a synchronized context.

notify(): wakes up only a single thread instead of multiple threads that are waiting on the objects monitor.

notifyAll(): wakes up all the threads.

Join() method is generally used to pause the execution of a current thread until another thread finishes its execution.

Notify:

Invoking the notify() method on an object wakesup a single thread that is waiting for the lock of this object.

On being notified, a waiting thread first transists to the blocked-for-lock-acquisition state to acquire the lock on the object, and not directly to the Ready to run state.

**Thread priorities** :

Thread are assigned priorities that the thread scheduler can use to determine how the threads will be scheduled.

Priorities are integer values from 1 (lowest priority given by the constant Thread.MIN\_PRIORITY) to 10(highest priority given by the constant Thread.MAX\_PRIORITY). The default priority is 5(Thread.NORM\_PRIORITY).

The priority of a thread can be set using the” setPriority()” and read using “getPriority()”.

**Thread scheduler:**

Scheduler in JVM implementations usually employ one of the two following strategies.

1. Preemptive scheduling.

If a thread has a higher priority than current running thread, It moves to the current running thread to ready to run state to execute this thread.

1. Time sliced or round-robin scheduling.

Where a running thread is allowed to run for a fixed length of time, after that it moves to ready-to-run state and waits for its turn to run again.

**What is reflection in java?**

Reflection is a powerful feature in java that allows a programmer to examine or modify the structure of a class at run time. That means a program can inspect and manipulate its own code, making reflection a useful tool for run time code generation, testing and more.

**Locking mechanism in java**:

**Collection hierarchy:**

**A diagram of a company

Description automatically generated**

If we want to represent a group of individual objects as a single entity, then we should go for collection.

Collection interface defines most common methods which are applicable for any collection object.

There is no concrete class which implements collection interface directly.

**LIST:**

If we want to represent a group of individual objects as a single entity where duplicates are allowed, and insertion is preserved then we should go for list.

**Array List**:

Underlying data structure is resizable or growable array. (it will resize 50% of its original size)

It is Non synchronized.

Heterogenous objects are allowed (except Tree set and Tree map everywhere heterogeneous objects are allowed).

Array list and vector implements Random access interface. So that we can access any element with same speed.

Array list is best choice if our frequent operation is retrieval (because it implements RAI).

Array list is the worst choice if our frequent operation is insertion or deletion in middle (Because several shift operations is required)

**Linked list:**

Implements List and Dequeue interfaces.

The underlying data structure is doubly linked list.

Note: Doubly linked list is a type of linked list in which nodes contains information and two pointers i.e., left pointer which points to previous node and right pointer which points to next node.

Does not support accessing elements randomly.

If our frequent operation is insertion or deletion, then it is the best choice.

**Vector:**

Underlying data structure for the vector is resizable or growable array. (it doubles the array size)

It is thread safe.

Implements serializable, cloneable and Random-access interface.

Best choice if our frequent operation is retrieval.

**Stack:**

Stack is a child class of vector.

It is specially designed for LIFO.

The elements are added as well as removed from the rear end.

It implements serializable, cloneable and Random-access interface.

**SET:**

If we want to represent a group of individual objects as a single entity, where duplicates are not allowed, and insertion is not preserved then we should go for set.

Doesn’t define an order for the elements hence index-based search is not supported.

**HashSet:**

The underlying data structure is hash table.

Duplicates are not allowed. If we try to insert any duplicates. We won’t get any compile or run time error, add method will simply returns false.

Only one null element can be added.

Hash set is the best choice if our frequent operation is search operation.

**Linked hash set:**

Child class of hash set.

The underlying data structure is hash table + doubly linked list.

It is an ordered version of hash set.

Linked hash set is the best choice to develop cache-based applications. Where duplicates are not allowed, and insertion order is preserved.

Sorted Set:

If we want to represent group of individual objects as a single entity, according to some sorting order where duplicates are not allowed then we should go for sorted set.

Note: All the elements of a sorted must implements comparable interface.

Since sorted is an interface objects cannot be created of type sorted set. We always need a class which extends this interface in order to create an object.

Tree set is the class which implements the sorted set.

Duplicates and null elements are not allowed in sorted set. For every element the elements are compared and sorted in ascending order.

**Tree set:**

The underlying data structure is balanced tree.

Duplicate objects are not allowed, and insertion order is not preserved, but all objects will be stored in some sorting order.

Heterogenous objects are not allowed. If we try to insert, we will get runtime exception called class cast exception.

Null insertion is allowed but only once.

Note: if we insert null to non-empty tree set then it will compare it with existing elements and will throw null pointer exception.

If we add null to an empty tree set it will accept, but after that if we insert any element it will through null pointer exception.

**QUEUE:**

If we want to represent group of individual objects prior to processing, then we should go for queue.

Follows FIFO approach.

Elements adds at rear end and removes from the front end.

**Priority queue:(class)**

If we want to represent a group of individual objects prior to processing according to some priority, then we should go for priority queue.

High priority element can be served first before a low priority element.

Insertion order is not preserved, and it is based on some priority.

Duplicate objects are not allowed. If we are depending on default natural sorting order compulsory objects should be homogenous and comparable otherwise, we will get runtime exception called class cast exception.

If we are defining our own sorting, then objects need not be homogenous and comparable.

Null is not allowed even as the first element also.

**Deque:(interface)**

Double ended queue that supports the addition and removal of elements from either end of the data structure. It can either be used as a queue (FIFO) or stack LIFO.

Since dequeue is an interface objects cannot be created of the type dequeue, we always need a class in order to create objects.

Array dequeue:(class)

Array dequeue implements dequeue interface.

It provides us a way to apply resizable or growable array.

Elements can be added or removed from both ends.

Array deque is faster than stack and queue.

**MAP:**

If we want to represent group of individual objects as key value pairs then we should go for map.

It can only contain unique key and can have duplicate values.

**Hash map:**

The underlying data structure is Hash table.

Insertion order is not preserved, and it is based on hash code of keys.

Duplicate keys are not allowed but values can be duplicated.

Only one null key is allowed but any number null values is allowed.

Heterogeneous objects are allowed for both key and values.

Hash map is the best choice if our frequent operation is search operation.

**Hash table:(class)**

It is synchronized in nature.

It doesn’t allow any null key or value.

**Linked hash set:**

Linked hash set is the child class of hash map.

Underlying data structure is hash table and Linked List

It is exactly same as hash map, but insertion order is preserved.

Sorted set:(interface)

Entries are maintained in an ascending order.

Implicitly implements the Red-black tree implementation.

Cannot store any null key.

**Java 8 features:**

Map ():

Map operation is used to transform each element of a stream into another object using a given function. It returns the new stream containing the transformed elements in the same order as the original stream.

This transformation is one-to-one, meaning each input element produces exactly one output element. So, if there are N elements in the stream, the map () operation will produce a new stream of N elements.

**FlatMap**():

It is the combination of a map and a flat operation.

This means we first apply the map function and then flatten the result.

The key difference is the function used by map operation returns a stream of values or a list of values rather than a single value, that’s why we need flattening, when we flat a stream of stream, it gets converted into stream of values.

To understand what flattening a stream consists in, consider a structure like [ [1,2,3], [4,5,6], [7,8,9]] which has “two levels”. It’s basically a big list containing three more list. Flattening this means transforming it in a “one level” structure e.g, [1,2,3,4,5,6,7,8,9] i.e, just one list.

**FILE OPERATIONS:**

Before understanding file operations, it is required that we have knowledge of Stream and file methods.

Stream:

A series of data is referred as Stream. In java, Stream is classified into two types, i.e., Byte Stream and Character Stream.

A diagram of a computer program

Description automatically generated with medium confidence

Byte stream:

Byte stream is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provides and executed with the byte data.

Character Stream:

Character stream is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

Java file class methods:

1. canRead():

The return type is Boolean.

The canRead() method is used to check whether we can read the data of the file or not.

1. createNewFile():

The return type is Boolean.

The createNewMethod() is used to create a new empty file.

1. canWrite():

The return type is Boolean.

This method is used to check whether we can write the data into the file or not.

1. exists ():

The return type is Boolean.

This method is used to check whether the specified file is present or not.

1. Delete ():

The return type is Boolean.

The delete () method is used to delete a file.

1. getName():

return type is string.

The getName() method is used to find the file name.

1. getAbsolutePath():
2. length ():

The length () method is used to get the size of the file in bytes.

1. list ():

The list() method is used to get an array of the files available in the directory.

1. mkdir()

return type is Boolean.

The mkdir() method is used for creating a new directory.

We can perform following operations on a file:

Create a file.

File f0= new File(”D:FileOperationExample.txt");

Get file information.

System.out.println(F0.getName());

Write to a file.

FileWriter fWrite= new FileWriter("D:FileOperationExample.txt");

fWrite.write("A named location used to store related information is referred to as a File.");

fWrite.close();

Read from a file.

File f1= new File("D:FileOperationExample.txt");

Scanner dataReader= new Scanner(f1);

While(dataReader.hasNextLine()){

String fileData= dataReader.nextLine();

System.out.println(fileData);

}

dataReader.close();

Delete a file.

 File f0 = **new** File("D:FileOperationExample.txt");

F0. Delete ();