**18-09-2025**

Synchronizing Code

o Synchronization and Locks

o Thread Deadlock yes

o Thread Interaction

o Using notifyAll( ) When Many Threads May Be Waiting yes

o Thread Executor yes

o ExecutorService,

o ScheduledExecutorService,

o ThreadFactory, and Callable

Try today to run the program

create 2 runnable task :

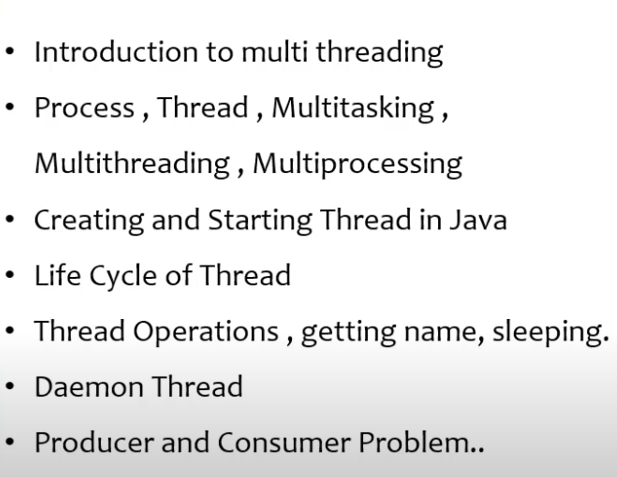
 1 : to print even nos in the range : start end

2 :  to print odd nos in the range : start end

execute these runnable task concurrently

**MULTITHREADING**

Multithreading Notes and details



**205.What is thread? what is multithreading?**

**Thread** are meant for **multi-tasking**.

Process of executing **multiple thread** at a time is called as **multithreading**.

Thread Concepts are not applicable if modules are linked with each other. Or if there is any dependency.

Main purpose of thread is to improve performance by reducing execution time.

**213.Demon thread**

demon thread low priority thread which runs on background to support non demon thread to execute.

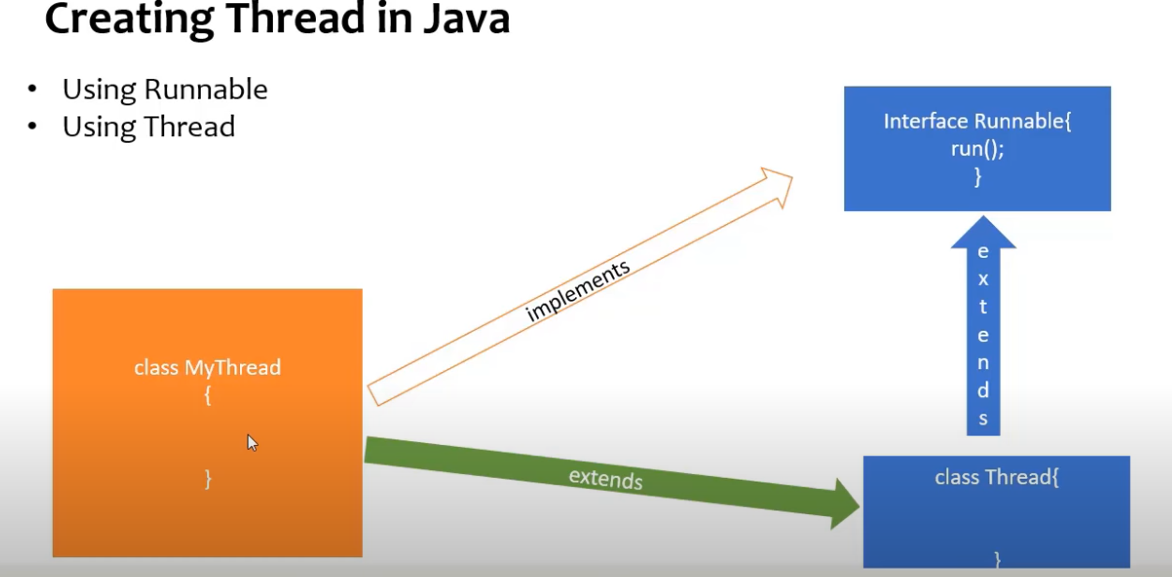
Daemon threads run in the background and perform tasks such as garbage collection.

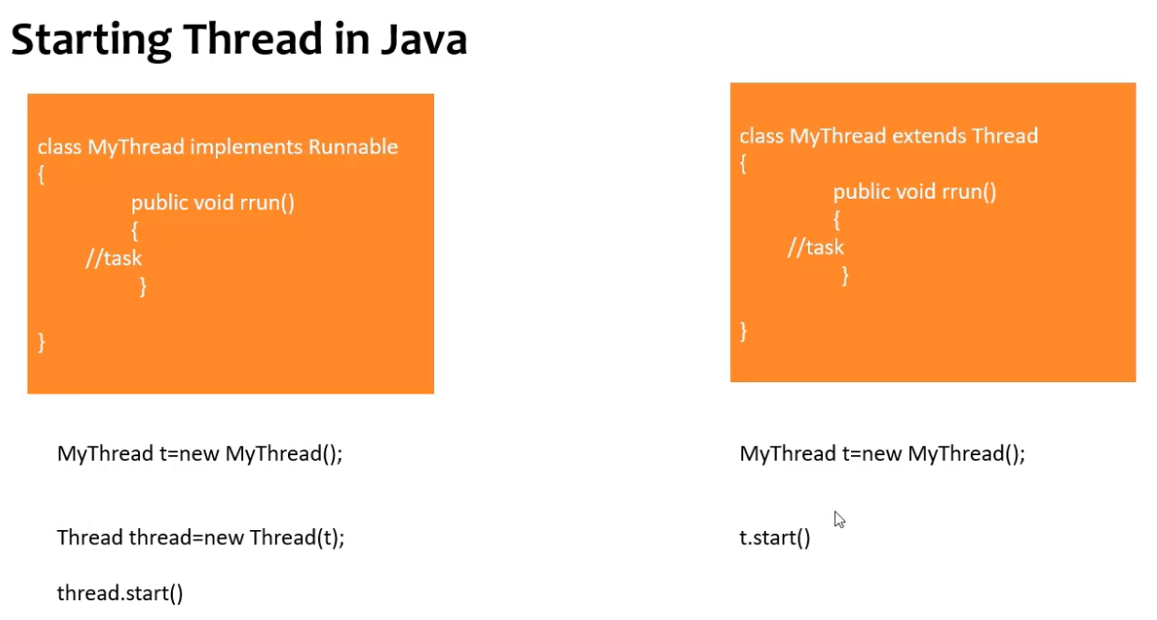
The JVM terminates when all user threads are done, even if daemon threads are still running.

A thread must be set as daemon before it starts, using set Daemon(true).

Example Use Cases: Logging, monitoring, or cleanup tasks.

**Creating a thread**

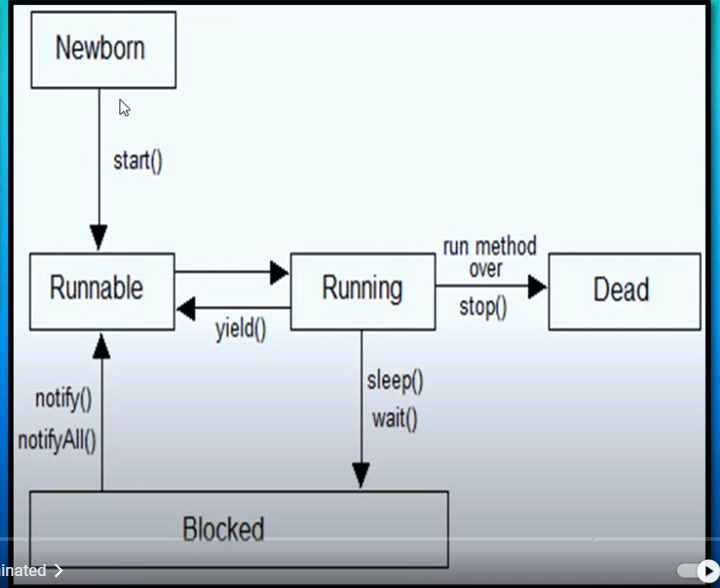
l

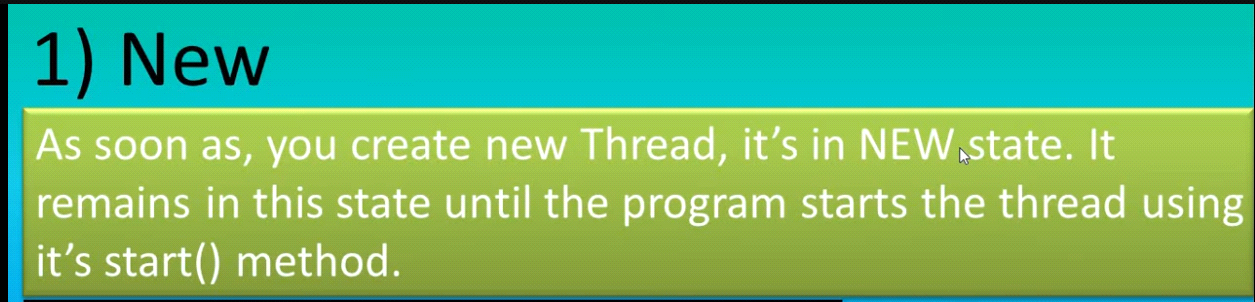


**Thread Life Cycle**

In Java, the **thread lifecycle** represents the various states a thread goes through from its creation to termination.

Java threads are managed by the JVM and can be in one of several states as defined in the Thread. State Enum. These states are:

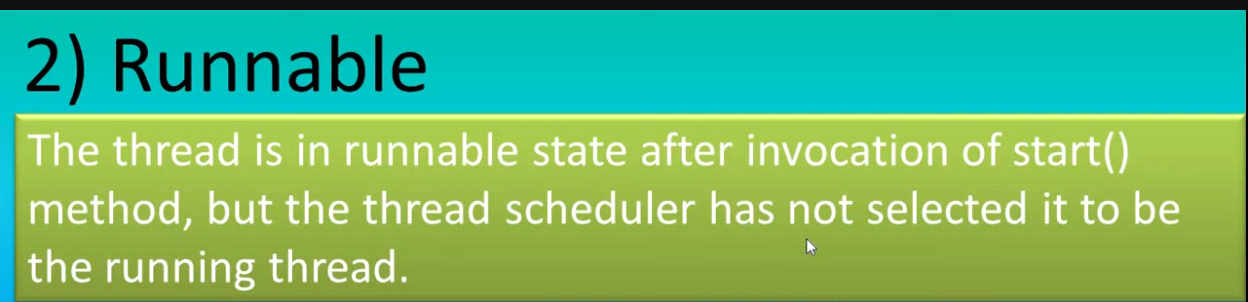




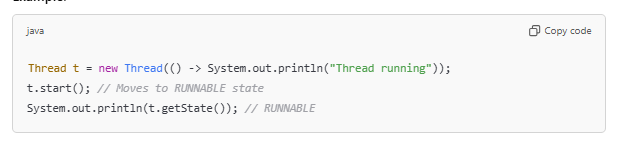


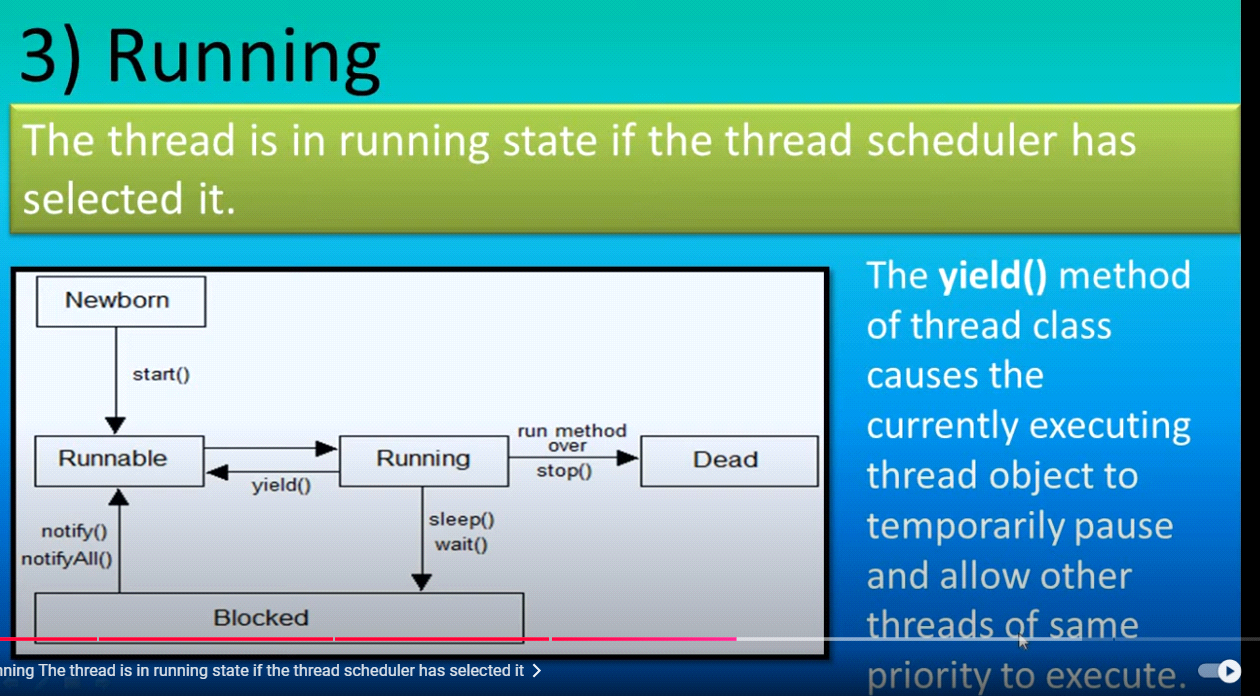
**2. Runnable**

* **State**: The thread is ready to run but may not be actively running. It is waiting for the CPU to schedule it.
* **Action**: The start() method is called, and the thread is eligible to run.
* **Transition**: The thread moves between RUNNABLE and RUNNING states depending on CPU scheduling



**Example-**





**Sleep()**

**Wait() in blocked**





**Key Methods Affecting Thread Lifecycle:**

* **start(): Moves thread from NEW to RUNNABLE.**
* **sleep(): Puts the thread in TIMED\_WAITING.**
* **wait(): Puts the thread in WAITING.**

**notify()**

* Wakes up **one single thread** waiting on the object's monitor.
* If multiple threads are waiting, only one (chosen arbitrarily by the JVM) will be notified.
* The notified thread competes with others for the lock after it is released by the current thread.

**notify All()**

* Wakes up **all threads** waiting on the object's monitor.
* All waiting threads will compete for the lock once it is released by the notifying thread.
* Useful when multiple threads need to proceed, but only one thread will actually acquire the lock
* **join(): Causes the current thread to wait until another thread completes.**

**Process to start thread**



**2nd process from thread class**



**215.Process to create thread and which is best and why**

A screenshot of a computer program

AI-generated content may be incorrect.

**where thread are used**

Threads are used in programming to achieve multitasking and improve performance by allowing concurrent execution of multiple parts of a program. Here are some common use cases where threads are utilized:

**Real-Time Applications**

**Purpose: To ensure timely execution of critical tasks.**

**Example:**

**A flight simulation program may use threads to update the simulation environment, track user inputs, and calculate trajectories.**

Difference bewen wait and sleep

Gaming

Purpose: To handle multiple tasks like game logic, rendering, and user input simultaneously.

Example:

One thread processes game physics, another handles rendering graphics, and another listens for user input.

**Key Differences:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **notify()** | **notifyAll()** |
| **Number of Threads** | Wakes up one thread only | Wakes up all waiting threads |
| **Usage** | Suitable when only one thread needs to proceed | Suitable when all waiting threads may need to proceed |
| **Lock Acquisition** | Notified thread competes for the lock | All notified threads compete for the lock |

**DRAWBACK OF THREAD:**

**Drawback of Thread:**

* When two threads are operating on a common data then Data corruption will happen. So to prevent data corruption we apply Thread Synchronization and this Thread Synchronization is the drawback.
* We use threads for better performance but When we apply thread synchronization to avoid data corruption, here **it slows down the execution time** as one thread is kept in wait status and the other threads keeps on executing.
* When two or multiple people is trying to book a ticket at same time, then it might be same ticket will booked for two or multiple persons.
* For thread we **can‟t set priority** which thread will run first.

***212. Thread synchronization*** *:*

* When two threads are working with common data concurrently, there is a risk of data corruption due to multitasking.
* We use the „synchronized‟ keyword to control. This keyword is applied to a method or a block of code, making sure that only one thread can execute that block at a time.
* In a synchronized block, when a thread acquires the lock (permission to execute), it can exclusively operate on the shared data. Other threads that attempt to access the synchronized block have to wait until the first thread releases the lock. Once the lock is released, one of the waiting threads is granted the lock, and it can then execute the synchronized block. This sequential access helps prevent data corruption and ensures the orderly execution of threads.

***What is join:***

* When a thread performs a task and pauses, another thread can pick up the work from where the first thread left off. This cooperative working approach is known as 'joining' threads. Thread 1 works until a certain point, then Thread 2 takes over from there. When Thread 2 pauses, Thread 1 resumes its work from the last point. This cycle continues, allowing threads to collaboratively complete a task.

**206.THREAD POOL**

**Thread Pool**

* A thread pool is a collection of threads. As soon as the pool has an idle thread, a task is assigned to one of them and executed.
* Thread pools are useful when you need to limit the number of threads running in your application simultaneously,
* It improves overall performance.
* Instead of creating a new thread for each task, tasks can be assigned to a thread pool for execution.

**Continuation:**

* Thread pools are often used in servers.
* Each connection arriving at the server via the network that is wrapped as a task and passed on to a thread pool.
* The threads in the thread pool concurrently process requests from the connections.

**Thread Safe? How to achieve it?**

In Java, **thread safety** refers to the ability of a program, method, or class to be safely used by multiple threads at the same time without causing any issues like data corruption or inconsistent behavior.

When multiple threads (or tasks) access the same resources (like variables, data structures, or files) concurrently, thread safety ensures that these threads don’t interfere with each other in a harmful way.

**Key Concepts of Thread Safety**

* **Shared Resources**:  
  Multiple threads might try to read or modify the same resource (like a variable or object) at the same time. Without proper control, this can lead to problems like **race** conditions or inconsistent states.
* **Race Conditions**:  
  A race condition occurs when multiple threads access shared data at the same time, and the result depends on the timing of thread execution. This can cause errors or unexpected behavior.
* **Critical Section**:  
  A critical section is a part of the code where shared resources are accessed or modified. It's important to ensure that only one thread can execute this section at a time to avoid conflicts.

**Ways to Achieve Thread Safety in Java**

* **Synchronization**:
* You can use the synchronized keyword to make sure that only one thread can execute a particular piece of code at a time.
* This can be applied to methods or specific blocks of code.

**Example**:

java

Copy code

public class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

public synchronized int getCount() {

return count;

}

}

* The synchronized keyword ensures that when one thread is executing the increment() method, no other thread can execute it simultaneously.
* **Volatile Keyword**:
* The volatile keyword ensures that changes made by one thread to a variable are immediately visible to all other threads.
* This is useful for simple flags or variables.

**Example**:

java

Copy code

private volatile boolean flag = false;

public void setFlagTrue() {

flag = true; // Visible to all threads immediately

}

* **Atomic Classes**:
* Java provides **atomic classes** in java.util.concurrent.atomic (like AtomicInteger, AtomicBoolean, etc.) that perform thread-safe operations without the need for synchronization.
* These classes use low-level concurrency mechanisms like Compare-And-Swap (CAS) to safely update variables.

**Example**:

java

Copy code

import java.util.concurrent.atomic.AtomicInteger;

public class AtomicCounter {

private AtomicInteger count = new AtomicInteger(0);

public void increment() {

count.incrementAndGet(); // Atomically increments the value

}

public int getCount() {

return count.get();

}

}

* **Thread-Safe Collections**:
* Java provides thread-safe collections like ConcurrentHashMap, CopyOnWriteArrayList, etc., which handle concurrency internally and allow multiple threads to access them safely.

**Example**:

java

Copy code

import java.util.concurrent.ConcurrentHashMap;

public class ThreadSafeMap {

private ConcurrentHashMap<String, Integer> map = new ConcurrentHashMap<>();

public void putValue(String key, Integer value) {

map.put(key, value); // Thread-safe operation

}

}

**208. Difference between thread and process**

|  |  |  |
| --- | --- | --- |
| Feature | Process | Thread |
| definition | A heavyweight, independent execution unit with its own memory space. | A lightweight, smaller execution unit within a process. |
| memory | Separate memory space | Shared memory with in a process. |
| Communication | Slower(inter process communication ) | Faster (shared memory used so faster. |
| overhead | Higher . | Lower. |
| dependency | Independent of other process | Dependent on other threads in same process. |
| creation | Required more resource | Faster because of light weight |
| Use case | Running independent programs | Concurrent tasks with in same program. |

**209. What is thread join?**

**A black background with white text

AI-generated content may be incorrect.**

In Java, the Thread.join() method is used to ensure that the current thread waits until the thread on which join() is called has completed its execution. It is particularly helpful when you want one thread to complete before the current thread continues its execution.

**Key Points:**

1. **Purpose:** To make the current thread wait for another thread to finish.
2. **Signature:**
   * void join(): Waits indefinitely for the thread to finish.
   * void join(long millis): Waits for the thread to finish for a maximum of the specified milliseconds.
   * void join(long millis, int nanos): Waits for the thread to finish for a maximum of the specified milliseconds and nanoseconds.

**Exception:** It throws InterruptedException if the current thread is interrupted while waiting.

**214. Difference between thread join and thread sleep method**

The Thread.sleep method pauses the execution of the current thread for a specified amount of time. It is commonly used to introduce delays in execution or simulate time-consuming tasks like loading, waiting for external resources, or pacing an application.

 **Static Method**: It is a static method of the Thread class, so it always pauses the currently executing thread.

 **Duration**:

* Accepts the duration in milliseconds.
* An optional nanoseconds argument can also be used.

 **Throws Exception**: Throws InterruptedException if the thread is interrupted while sleeping.

 **State**: The thread enters the TIMED\_WAITING state during sleep.

A screenshot of a computer program

AI-generated content may be incorrect.

**Thread.join()**

If we **want to wait** for a thread to finish its task **before moving on**. That’s where join() comes in.

Useful when you want threads to run sequentially.

**216. What is thread scheduler**

**A screenshot of a computer

AI-generated content may be incorrect.**

**217. Explain about priority of thread?**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**217.Explain about priority of thread?**

Threads with higher priority are typically executed before threads with lower priority.

However, thread priority is not a guarantee of execution order since it is dependent on the **thread scheduler** implementation, which is platform-dependent.

A screenshot of a computer code

AI-generated content may be incorrect.

**218. What are executor services**

**ExecutorService is part of Java’s concurrency framework. It's used to manage a pool of threads to run tasks asynchronously without manually creating and starting threads.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A white text with black text

AI-generated content may be incorrect.**

**220.Advantage and Disadvantage of thread**

A table with text on it

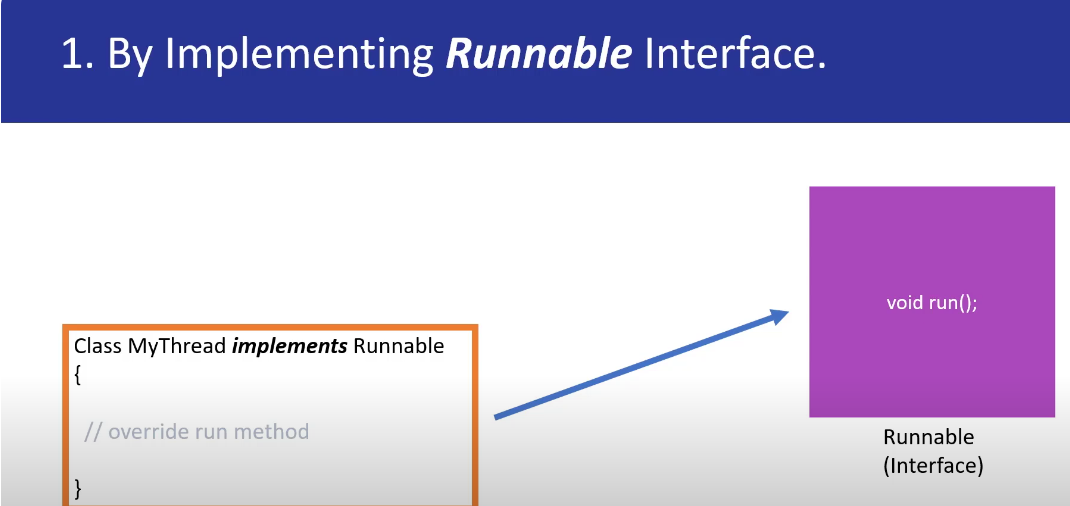
AI-generated content may be incorrect.

**224. Define Threads and its types/ Define Threads and its types/** Ways to create thread?

**A blue and white background with black text

AI-generated content may be incorrect.**

<https://www.youtube.com/watch?v=tHgCt6TRrWA>



A screen shot of a diagram

AI-generated content may be incorrect.

Thread1

A screenshot of a computer

AI-generated content may be incorrect.

Thread-2

A screenshot of a computer

AI-generated content may be incorrect.

**Which is recommended?**

**233.What is object lock vs class lock**

**Object Lock**

A lock associated with a specific instance of a class.

Ensures that only one thread can access synchronized instance methods or blocks of the same object at a time.

**How It Works**:

When a thread acquires an object lock, other threads are prevented from executing any synchronized instance method/block of that particular object.Each object has its own lock.



**Class Lock**

* **Definition**: A lock associated with the .class object of a class, rather than a specific instance.
* **Purpose**: Ensures that only one thread can access synchronized **static methods/blocks** of a class at a time.
* **How It Works**:
  + When a thread acquires a class lock, it prevents other threads from executing any synchronized **static method/block** of the same class.
  + Class locks are associated with the **class object**, not instances.

A screenshot of a computer program

AI-generated content may be incorrect.

**Can we call run method before start method**

**Yes, you can call the run() method directly before calling the start() method in Java, but it will not create a new thread. Here's a detailed explanation:**

**1. Calling run() Directly**

* **The run() method is just a regular method in the Thread class or your custom class implementing Runnable.**
* **If you call run() directly, it will execute on the current thread instead of creating a new thread**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

What is deadlock and how to avoid it?

**A black background with white text

AI-generated content may be incorrect.+**

**When dead lock happens**

**A white paper with black text

AI-generated content may be incorrect.**

**What is thread joiner?**

**How to avoid it?**

**What is thread safe? Benefits of thread safe.**

A piece of code, object, or class is considered thread-safe if it functions correctly when accessed or modified by multiple threads concurrently, without causing any race conditions, inconsistent data, or unexpected behavior.

Thread safety ensures that shared resources (e.g., variables, data structures) are accessed in a controlled and predictable manner in multithreaded applications.

**A screenshot of a computer program

AI-generated content may be incorrect.**

**How to make 2 threads run one after another.**

To make two threads run **one after another** in Java, you can use **synchronization** techniques to ensure that one thread completes its execution before the other starts. Below are several ways to achieve this

A screenshot of a computer program

AI-generated content may be incorrect.