**MULTITHEREDING:**

NOTE: IF THERE IS NO DEPENDANCY THEN ONLY USE THREADING CONCEPT

-Read volatile and volatile vs synchronized diff.

Imagine you have two people (threads) working together on a puzzle. The puzzle pieces represent a variable in your program. If one person (thread) changes a piece, you want the other person (thread) to immediately see that change. Using **volatile** is like having a rule that says if one person moves a puzzle piece, everyone else must immediately see that change, so they don't work with outdated information.

So, in Java, you use **volatile** to ensure that variables are updated and shared correctly among threads, preventing bugs that can occur when different threads don't see the most current values.

# MULTITHEREDING IN JAVA IT IS A PROCESS OF EXECUTING MULTIPLE THREADS SIMULTANEOUSLY.

OR

#IT IS PROCESS OF EXECUTING MULTIPLE TASK SIMULTANEOUSLY WHERE EACH TASK IS SEPARATE INDEPENDENT PART OF THE SAME PROCESS/PROGRAM.

**MULTIPROCESSING:**

# IT IS PROCESS OF EXECUTING MULTIPLE TASK SIMULTANEOUSLY WHERE EACH TASK IS SEPARATE INDEPENDENT PROCESS (PROGRAM).

1. **Process**: A process is an independent instance of the JVM that runs a Java program. It has its own heap memory, classloading, and system resources. Each process can run one or more Java applications.
2. **Thread**: A thread is a smaller unit of a process responsible for executing tasks. Threads within the same process share the same memory space, which allows them to communicate and share data more efficiently than separate processes. Java applications can have multiple threads running concurrently to perform various tasks.

Here's a simplified analogy:

* A process is like a standalone house. Each house (process) has its own set of rooms (memory and resources) and can operate independently.
* A thread is like a person inside the house. Multiple people (threads) can be inside the same house (process), and they can collaborate by sharing the rooms (memory) and communicating with each other.

Top of Form

**#THREAD:**

-A THREAD IS A LIGHTWEIGHT SUB-PROCESS,

OR

-THE SMALLEST UNIT OF PROCESSING.

OR

-INDIPENDANT FLOW OF EXECUTION IS CALLED THREAD.

**-MULTIPROCESSING AND MULTITHEREDING, BOTH ARE USED TO ACHIEVE MULTITASKING.**

**#PROCESSING** :

A PROGRAM IS IN EXECUTION.

#HOWEVER, WE USE MULTITHEREDING THAN MULTIPROCESSING BECAUSE THREADS USE A SHARED MEMORY AREA

THEY DON'T ALLOCATE SEPARATE MEMORY AREA SO SAVES MEMORY, AND CONTEXT-SWITCHING BETWEEN THE

THREADS TAKES LESS TIME THAN PROCESS.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

1) Process-based Multitasking (Multiprocessing)

* Each process has an address in memory. In other words, each process allocates a separate memory area.
* A process is heavyweight.
* Cost of communication between the process is high.
* Switching from one process to another requires some time for saving and loading [registers](https://www.javatpoint.com/register-memory), memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

* Threads share the same address space.
* A thread is lightweight.
* Cost of communication between the thread is low.

#### **At least one process is required for each thread.**

**# THERE ARE TWO WAY OF EXECUTING THREAD.**

**-** Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

**1) BY EXTENDING THREAD CLASS.**

**2) BY IMPLEMENTING RUNNABLE INTERFACE.**

**-Runnable interface has only one method which is *public abstract void run();***

@ THREAD CLASS CONSTRUCTORS

1)THREAD T=NEW THREAD();

2)THREAD T=NEW THREAD(STRING NAME);

3)THREAD T=NEW THREAD(RUNNABLE R);

4)THREAD T=NEW THREAD(STRING NAME,RUNNABLE R);

5)THREAD T=NEW THREAD(THREADGROUP G,RUNNABLE R);

6)THREAD T=NEW THREAD(THREADGROUP G,STRING NAME);

7)THREAD T=NEW THREAD(THREADGROUP G,STRING NAME,RUNNABLE R);

8)THREAD T=NEW THREAD(THREADGROUP G,STRING NAME,RUNNABLE R,LONG STATIC SIZE);

# **DIFFERENCE BETWEEN T. START() AND T.RUN();**

@ T.START()

-IF WE CALLING T.RUN METHOD DIRECT OUR MAIN THREAD IS RESPONSIBLE TO START EXECUTING THE RUN METHOD SO WE DON'T GET The FACILITY OF MULTIPLE THREADING.

-BUT IF WEE CALL START METHOD THEN MAIN METHOD CHECK WHETHER WE HAVE START METHOD IF NOT IT WILL CHECK ITS PARENT CLASS AND PARENT CLASS CONTAIN THE START METHOD WHICH IS INTERNALLY CALL THE RUN METHOD SO WE GET THE FACILITY OF MULTIPLE THREAD BECZ THE CHILD THREAD METHOD IS RESPONSIBLE TO EXECUTE THE RUN METHOD NOT main method JVM.

@ **START();**

-THE MAIN WORK OF THE START METHOD INSIDE THE THREAD CLASS

1)IT WILL REGISTER THIS NEW THREAD TO THREAD SCHEDULER

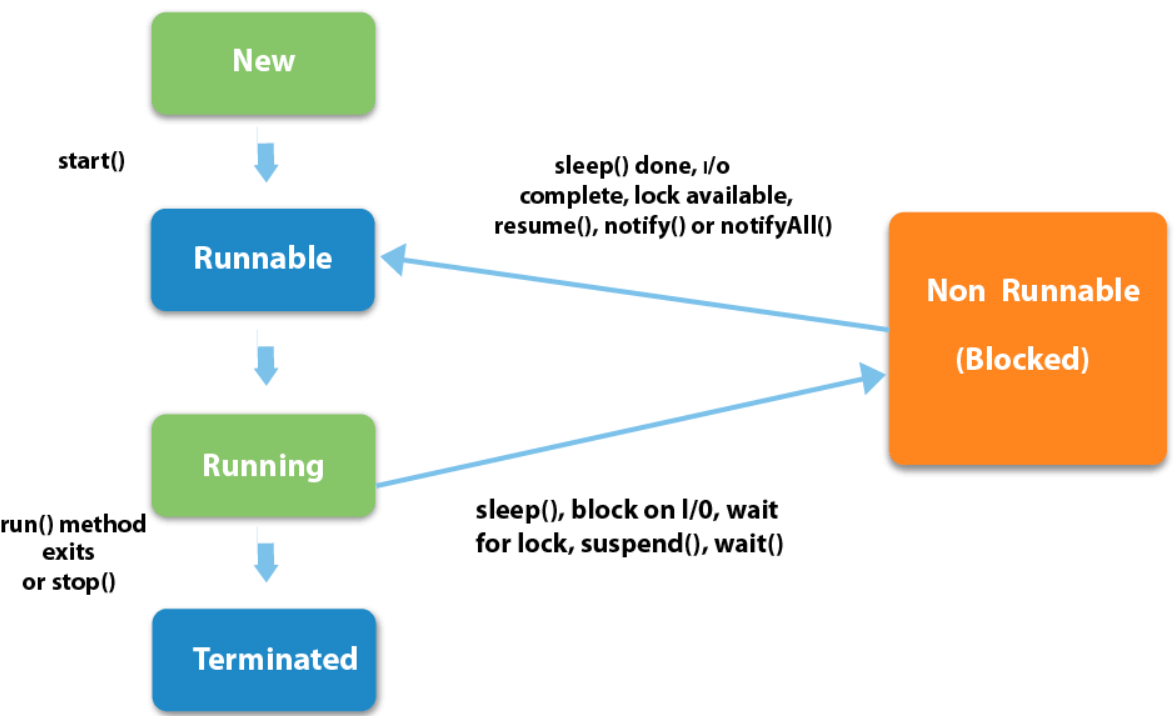
2)AND ALL PERFORMANCE AND MANDATORY ACTIVITY

3)INTERNALLY CALL RUN();

@ WE CAN OVERLOAD THE RUN METHOD BUT JVM ONLY CALL METHOD WITH NO ARGUMENT .WE HAVE TO EXPLICITLY CALL THAT OVERFLOW METHOD

-SO YOU HAVE TO CREATE THE OBJECT FOR THREAD CLASS TO RUN METHOD;

*THREAD LIFE CYCLE:*



**@ Thread Scheduler:**

- It a part of JVM which decide which thread should be execute first.

-Only one thread at a time can run in a single process.

-If multiple thread are waiting in non-running state to get the chance to execute its totally decided by thread scheduler but which get first execute we cant say becz which type scheduling algorithm there are using if differ from jvm to jvm

-The thread scheduler mainly uses pre-emptive or time slicing scheduling to schedule the threads.

@ we can not start Thread twice otherwise we will get RE:illigalThreadStateException .

@Out of two method implement Runnable interface is the best choice because in case of extends Thread class we allready extending one

class so we can not extend any other class so we do get the facility of multiple inheritance.

-Every thread in java having some name which is explicitly provided by programmer or default name given by jvm.

@**TO GET THREAD NAME THERE ARE SOME METHOD**

1)public final String getName()

2)public final void setName()

3)THREAD.CURRENTTHREAD()

@**THREAD PRIORITY**

-VALID RANGE OF PRIORITY IS 1-10;

-IF TWO THREAD HAVING SAME PRIORITY THEN WE CANT EXPECT EXACT EXECUTION ORDER IT DEPENDS ON THREAD SCHEDULER

-THE DEFAULT PRIORITY FOR MAIN IS 5 AND ALL OTHER THREAD CAN INHERIT FROM PARENT TO CHILD PRIORITY.

\*THERE ARE 3 METHOD WHICH CAN AVOID THE THREAD EXECUTION.

1) YIELD METHOD

public static native void yield();

(NATIVE METHOD)

-IF THREAD WANT TO PASS THE CURRENT EXECUTING THREAD TO GIVE THE CHANCE TO WAITING THREAD OF SAME PRIORITY(I.E CURRENT EXECUTING THREAD LEAVE OR

PAUSE ITS EXECUTION AND GIVE THE PROCESSOR TO THE OTHER REMAINING OTHER WAITING THREAD OF SAME PRIORITY)

-IF WAITING THREAD HAS LESS PRIORITY THEN CURRENT EXECUTING THREAD WILL CONTINUE ITS EXECUTION

-IF WAITING ALL THREAD HAS SAME PRIORITY THEN WE DON'T NOW WHICH THREAD GET THE CHANCE ITS TOTALLY DEPEND UPON THE JVM AND THREAD SCHEDULER

-IF THE METHOD IS YIELD THEN WE DON'T KNOW WHEN THE METHOD GET CHANCE AGAIN IT ALSO DEPEND UPON JVM.

-ITS LIKE PREMPTIVE SCHEDULING.

-Some operating system does not provide proper support for yield method.

2) JOIN METHOD

Public void join();throws IE

Public final native void join(long ms);IE

-IF A THREAD WANT TO WAIT UNTIL COMPLETING SOME OTHER THREAD THEN WE SHOULD GO FOR JOIN();

-IF A(THREAD) WANT TO WAIT FOR B(THREAD) THEN

-A THREAD CALLS B.JOIN SO A() METHOD GOING TO WAIT UNTIL B() COMPLETE .

-WHEN EVER WE ARE CALLING JOIN METHOD THERE MAY BE CHANCE OF INTERRUPTED EXCEPTION BECZ OTHER THREAD CAN INTERRUPT SO WHILE USING

JOIN METHOD ALWAYS USE EXCEPTION HANDLING AS IT IS CHECKED EXCEPTION.

**Interrupt, interrupted vs isInterrupted**

**-**it is used to check whether thread is interrupted or not.

-Interrupted ():it is first return whether your thread is interrupted then it return

True then it make interrupted status to true to false.

-isInterrupted(): it does not clear any status to true to false.

-interrupted():it chance the result if we use two time where isinterrupted does not do any thing if we run many times.

**Syntax**

**1)public static Boolean interrupted()**

**{**

**}**

**2)public Boolean isInterrupted()**

**{**

**}**

3) SLEEP()

WHEN A THREAD DO WANT TO PERFORM ANY OPERATION THEN IT GOING TO SLEEP .

-THREAD CAN BE INTERRUPTED BY THREAD CLASS INTERRUPT METHOD.

**1) SYNCHRONIZATION**

-IT IS USED TO REDUCE THE DATA IN CONSISTENCY PROBLEM.

-WHEN MULTIPLE THREAD TRYING TO OPERATE SIMULTANEOUSLY ON SAME JAVA OBJECT THEN THERE MAY BE CHANCE OF DATA IN CONSISTENCY.

-TO AVOID IT WE USE SYNCHRONIZED KEYWORD TO METHOD OR BLOCK.

-WHEN WE APPLY SYNCHRONIZED KEYWORD WITH METHOD THEN AT A TIME ONLY ONE THREAD IS ALLOWED TO EXECUTE THE METHOD WITH GIVEN OBJECT.

-SYNCHRONIZATION IS USED TO AVOID THE INTERFACE OF THE TWO THREAD.

-EVERY OBJECT IN JAVA HAS A UNIQUE LOCK BUT WHEN EVER WE USE SYNCHRONIZED CONCEPT THEN ONLY LOCK CONCEPT COME IN PICTURE.

-IF A THREAD WANT TO EXECUTE THE SYNCHRONIZED METHOD ON THE GIVEN OBJECT HE NEED LOCK OF THAT OBJECT WHICH IS PROVIDED BY JVM.

-BUT AT A TIME ONLY ONE THREAD CAN GET LOCK AND AFTER COMPLETION LOCK WILL RELEASE.

IMP

-IF A THREAD EXECUTING SYNC.METHOD ON GIVEN OBJECT THEN REMAINING THREAD ARE NOT ALLOWED TO EXECUTE ANY SYNCHRONIZED METHOD SIMULTANEOUSLY ON THAT SAME OBJECT BUT REMAINING THREAD ARE ALLOWED TO EXECUTE NON-SYNC.SIMULTANEOUSLY.

-EVERY OBJECT IN JAVA HAS TWO AREA SYNC. AREA AND NON-SYNC. AREA IN SYNC. AREA WE CAN PERFORM THE ADD, MODIFY, DELETE OPERATION WHICH IS ONLY APPLICABLE TO THOSE THREAD WHICH HAVING OBJECT LOCK.

-AND THE NON-SYNC. AREA CAN ACCESS ANY THREAD ONLY FOR READ OPERATION

-IF MULTIPLE THREAD OPERATING ON DIFFERENT OBJECT WE DON'T NEED SYNCHRONIZATION.

-WHEN WE USE STATIC SYNC. METHOD THEN WE GET REGULAR O/P BECZ STATIC BELONG TO CLASS NOT OBJECT HERE WE GET CLASS LEVEL LOCK AS CLASS LOCK.

OTHER THREAD HAS TO WAIT UNTIL THE FIRST THREAD COMPLETE HIS JOB.

2) SYNCHRONIZATION BLOCK

-If few line of code require the synchronisation then it is not recommended to

Enclose entire method to be synchronized.

-IF FEW LINES REQUIRE SYNC.SO ONLY THAT FEW LINE ENCLOSED IN THE SYNCHRONIZED BLOCK.

-WE USE SYNC (THIS)//FOR CURRENT OBJECT

-WE HAVE SYNC (B)//FOR OTHER OBJECT

-WE HAVE SYNC (CLASSNAME.CLASS)//FOR CLASS LEVEL LOCK

-LOCK CONCEPT APPLICABLE ONLY FOR OBJECT AND CLASS NOT FOR PRIMITIVE DATA TYPE.

-A THREAD CAN HAVE MULTIPLE LOCK ALSO.

**1) INTER THREAD COMMUNICATION.**

-TWO THREAD CAN COMMUNICATE WITH EACH OTHER BY WAIT (), NOTIFY (), NOTIFYALL () METHOD.

IMP

-THE THREAD WHICH IS EXPECTING UPDATION IS RESPONSIBLE TO CALL WAIT() METHOD THEN IMMEDIATELY THE THREAD WILL ENTER INTO WAITING STATE.

-THE THREAD WHICH IS RESPONSIBLE TO PERFORM UPDATION,AFTER PERFORMING UPDATION IT IS RESPONSIBLE TO CALL NOTIFY METHOD THEN WAITING THREAD WILL

GET THAT NOTIFICATION AND CONTINUE ITS EXECUTION WITH THOSE UPDATED ITEMS.

-WAIT,NOTIFY NOTIFYALL ARE AVAILABLE IN OBJECT CLASS BECZ IF THREAD CLASS WANT TO EXECUTE THIS METHOD ON ANY JAVA OBJECT.LIKE SQL UPDATION,

STUDENT CLASS UPDATION ETC

-TO CALL WAIT ,NOTIFY OR NOTIFYALL METHODS ON ANY OBJECT THREAD SHOULD BE OWNER OF THAT OBJECT I.E THE THREAD SHOULD HAS LOCK OF THAT OBJECT

I.E THE THREAD SHOULD BE INSIDE SYNC. AREA OR BLOCK.

-HENCE WE CAN CALL WAIT NOTIFY AND NOTIFYALL METHOD ONLY FROM SYNC.AREA. OTHER WISE WE WILL GET RE: SAYING **ILLIGALMONITERSTATEEXCEPTION**

-IF A THREAD CALLS WAIT METHOD ON ANY OBJECT IT IMMEDIATELY RELEASES LOCK OF THAT PARTICULAR OBJECT AND ENTER INTO WAITING STATE.

-EVERY WAIT METHOD THROWS INTERRUPTED EXCEPTION WHICH IS CHECKED EXCEPTION.

-IN THREAD LIFE CYCLE IF THREAD GO INTO WAITING STATE AFTER HE GETTING NOTIFICATION HE GOES INTO ANOTHER WAITING STATE TO GET

LOCK OF OBJECT AND THEN IT WILL GO INTO THE READY/RUNNABLE STATE AND THEN T.S GIVE PRIORITY.

\*\*\*PRODUCER COSUMER PROBLEM IS BEST EXAMPLE FOR THREAD COMMUNICATION

-HERE PRODUCER THREAD IS RESPONSIBLE TO PRODUCE ITEM IN THE QUEUE WHERE CONSUMER THREAD IS RESPONSIBLE TO CONSUME ITEM FROM QUEUE.

IF QUEUE IS EMPTY THEN CONSUMER GOES INTO WAITING STATE WHEN PRODUCER PRODUCE THE ITEM IT WILL GIVE NOTIFICATION TO CONSUMER AN THEN

CONSUMER CONSUMES ITEM.

@ DIFFERENCE BETWEEN NOTIFY VS NOTIFYALL

-WE CAN USE NOTIFY METHOD TO GIVE THE NOTIFICATION FOR ONLY ONE WAITING THREAD IF MULTIPLE THREAD ARE WAITING THEN ONLY ONE THREAD WILL NOTIFY AND REMAINING THREAD HAVE TO WAIT FOR FURTHER NOTIFICATION.WHICH THREAD WILL BE NOTIFIED WE CAN'T EXPECT IT DEPENDS ON JVM.

-WE CAN USE NOTIFYALL() TO GIVE THE NOTIFICATION FOR ALL WAITING THREAD OF A PARTICULAR OBJECT.EVEN THOUGH MULTIPLE THREAD NOTIFY BUT EXECUTION WILL BE PERFORMED ONE BY ONE BECZ THREAD REQUIRED LOCK OF OBJECT AND ONLY ONE LOCK IS AVAILABLE.

@ DEADLOCK

-DEADLOCK IS SITUATION WHERE ONE THREAD IS WAITING FOR OBJECT THAT IS ACQUIRE BY THE ANOTHER THREAD AND SECOND THREAD IS WAITING FOR OBJECT WHICH

IS ACQUIRE BY FIRST THREAD BOTH THE THREAD ARE WAITING FOR EACH OTHER TO RELEASE FOR INFINITE AMOUNT OF TIME LOCK THIS IS CALLED DEADLOCK.

-WE CAN NOT NOT SOLVE DEADLOCK CONDITION BUT THERE ARE SOME PREVENTION TECHNIQUE

1)ELIMINATE THE MUTUAL EXCLUSION.

2)ELIMINATE HE HOLD AND WAIT AND CIRCULAR WAIT

THE BEST PREVENTION FOR DEAD LOCK IS BANKERS ALGORITHM.

@ DAEMON THREAD

-DAEMON THREAD ARE NOTHING BUT SERVICE PROVIDER .THEY PROVIDE SERVICES TO MAIN THREAD OR USER THREAD.

-LIFE OF DAEMON THREAD IS DEPEND ON MAIN THREAD MERCY WHEN MAIN THREAD DIE GARBAGE COLLECTOR ELIMINATE THIS DAEMON THREAD

OR

-THE THREAD WHICH ARE EXECUTE AT BACKGROUND CALLED DAEMON THREAD.

EX.GARBAGE COLLECTOR ,SIGNAL DISPATCHER.

-THE MAIN OBJECTIVE OF DAEMON THREAD IS TO PROVIDE SERVICES OR SUPPORT TO NO-DAEMON THREAD(MAIN THREAD).

-USUALLY DAEMON THREAD HAS LOW PRIORITY BUT THEY CAN RUN IN HAVE HIGH PRIORITY ALSO.

TWO METHODS IN DAEMON THREAD

1)PUBLIC BOOLEAN ISDAEMON()

2)PUBLIC VOID SETDAEMON(BOOLEAN B);

-WE CAN CHANGE ANY THREAD AS DAEMON THREAD BUT BEFORE THE STARTING OF THREAD ONLY OTHERWISE WE GET

-ILLIGALTHREADSTATEEXCEPTION

-MAIN THREAD ALWAYS NON DAEMON AND ALL OTHER THREAD WILL INHERIT FOR PARENT TO CHILD

-Read volatile and volatile vs synchronized diff.