Day 13 – Concurrency in JAVA

**🕒 Training Agenda**

**Part 1: Introduction & Foundations**

* Why concurrency? Performance, responsiveness, scalability.
* Difference between **parallelism vs concurrency**.
* Threads in Java (vs Python threading model, GIL concept → contrast).
* Java Memory Model (happens-before, visibility, atomicity).
* Example: Simple Thread creation and execution.

**Part 2: Core Concurrency in Java**

* Runnable and Callable.
* ExecutorService and Thread Pools.
* Future and CompletableFuture.
* Synchronization (synchronized, volatile, ReentrantLock).
* Example Programs:
  + Bank Account deposit/withdraw race condition.
  + Fixing with synchronized or Lock.

**Part 3: Real-Life Use Case**

**Use Case: E-commerce Order Processing System**

* Tasks:
  + Validate Order
  + Check Inventory
  + Process Payment
  + Send Notification
* Parallelize with CompletableFuture.supplyAsync() and combine results.
* Show improvement over sequential processing.

**Part 4: Advanced Concurrency & Best Practices**

* ForkJoinPool & parallel streams.
* Producer-Consumer with BlockingQueue.
* Handling Deadlocks (example + detection strategies).
* Executors vs Virtual Threads (Project Loom preview).

**Part 5: Mini Sample Application**

**App:** Multithreaded Log Analyzer

* Input: Large log file with millions of entries.
* Tasks split across threads:
  + Count errors,
  + Extract unique users,
  + Aggregate request latency.
* Merge results at the end (using ExecutorService).

This resonates with DB/Python folks (like Spark or multiprocessing).

**Part 6: Wrap-Up & Q/A**

* Best practices (thread safety, immutability, avoiding shared state).
* Compare Java concurrency with Python asyncio/multiprocessing.
* Share GitHub repo + further reading.

**📂 GitHub Repo Structure (for participants)**

java-concurrency-training/

│── basics/

│ ├── HelloThread.java

│ ├── BankAccountRace.java

│ ├── BankAccountSync.java

│

│── executor/

│ ├── ExecutorExample.java

│ ├── CompletableFutureDemo.java

│

│── realusecase/

│ ├── OrderProcessingSystem.java

│

│── advanced/

│ ├── ForkJoinExample.java

│ ├── ProducerConsumerQueue.java

│ ├── DeadlockExample.java

│

│── sampleapp/

│ ├── LogAnalyzer.java

│ └── logs/sample.log

│

└── README.md (with instructions, agenda, exercises)

Parallel Processing.

Sync vs Async Operations.

Threads in JAVA.

Multiple-Threads Concepts.

Thread Group.

Thread Life Cycle.

Different ways of Creating Thread.

Thread Pool.

Day 12 Revisit – Date & Time API in JAVA

Date class from java.util & java.sql package.

Util package Date will provide complete Date, time and time zone details.

SQL package Date will provide Date, Month & Year details only.

Locale, LocalDateTime, Time

Converting one format to another format.

SimpleDateFormat

Parse and pattern methods.

Deprecated Methods and avoiding deprecation warning.

Parallel Operation – (6 task)

1. Print Hello World in Console (10 ms)
2. Open a File and display it’s content (450 ms)
3. Get input from user and display it in a given format. (12 sec)
4. Open a DB table and fetch all it’s content and display it in console. (800 ms)
5. Send a data through a network (750 ms)
6. Process a excel sheet content. (450 ms)

Program – Named sequence of execution or operation. (Sync processing – Step-by-step or sequential processing) serial processing [one after another]

1. One operation at a time.
2. Finish the first operation & go to second operation

Multi-Core Processors – (Penta core, octa core, hexa core processors) – 3.65 GHz.

Async or Parallel processing (Multi-Thread) [all at a time- using multi-core and multi-thread] -- Concurrent Operation [Concurrency]

Thread – Is a lite weight process.

Process – is a heavy weight Thread.

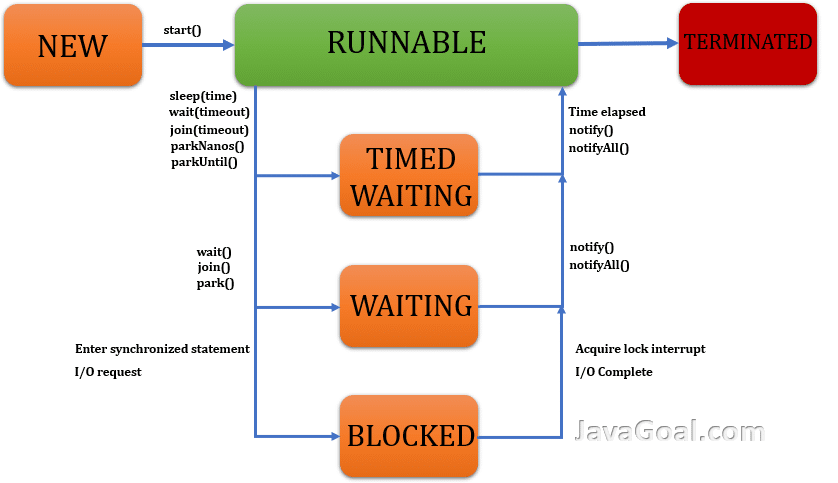
Process is always associated with process id (pid).

Process – Program during execution time.

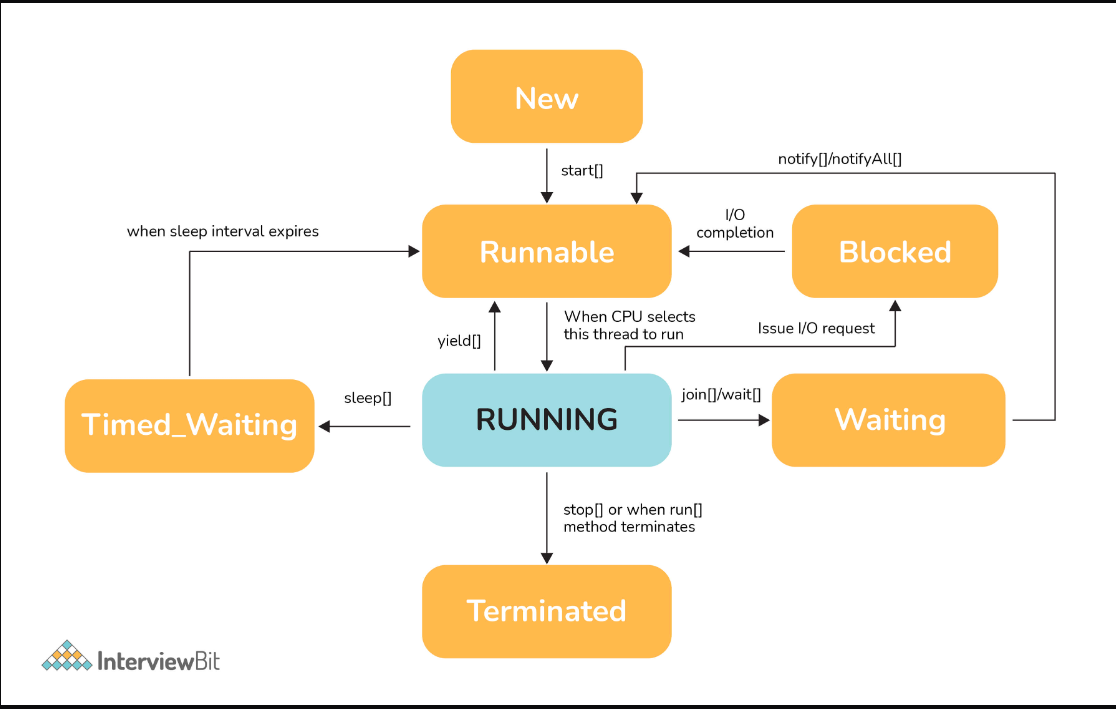
WinWord is the name of the program (WinWord.exe) – WinWord.exe creates a process when it is running (It has a PID)

1. Spell-Check Thread
2. Grammer-Check Thread
3. Auto-Save Thread

Creating a Thread in JAVA



Thread Life Cycle.



Various States of Thread. NEW, RUNNABLE, RUNNING, BLOCKED/WAITING/SLEEPING (NON-RUNNING or BLOCKED) , TERMINATED.

A thread state. A thread can be in one of the following states:

* [NEW](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#NEW)  
  A thread that has not yet started is in this state.
* [RUNNABLE](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#RUNNABLE)  
  A thread executing in the Java virtual machine is in this state.
* [BLOCKED](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#BLOCKED)  
  A thread that is blocked waiting for a monitor lock is in this state.
* [WAITING](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#WAITING)  
  A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
* [TIMED\_WAITING](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#TIMED_WAITING)  
  A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
* [TERMINATED](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#TERMINATED)  
  A thread that has exited is in this state.

Ways of Creating Thread (Using the Default package – java.lang )

1. Using Thread Class
2. Using Runnable Interface. (Recommended)

Thread Properties

Id,

Name

Priority (int) [1-10] 1 – MinPriority, 5 – Normal Priority, 10-Max Priority

Status (Enum)

isAlive

isDaemon

isVirtual

isTerminated

**package** com.example;

**import** java.util.Set;

**public** **class** ThreadDemo {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

Thread t1 = **new** Thread();

Thread t2 = **new** Thread("MyThread");

System.***out***.println("Thread Properties");

System.***out***.println("T1 ID:"+t1.~~getId~~());

System.***out***.println("T2 ID:"+t2.~~getId~~());

System.***out***.println("T1 Name:"+t1.getName());

System.***out***.println("T2 Name:"+t2.getName());

t2.setPriority(8);

System.***out***.println("T1 Priority:"+t1.getPriority());

System.***out***.println("T2 Priority:"+t2.getPriority());

System.***out***.println("Min Priority:" + Thread.***MIN\_PRIORITY***);

System.***out***.println("Norm Priority:" + Thread.***NORM\_PRIORITY***);

System.***out***.println("Max Priority:" + Thread.***MAX\_PRIORITY***);

System.***out***.println("T1 State :"+t1.getState());

System.***out***.println("T2 State :"+t2.getState());

// Get all active threads

Set<Thread> threads = Thread.*getAllStackTraces*().keySet();

System.***out***.println("Currently running threads:");

System.***out***.println("---------------------------------------------------------");

System.***out***.printf("%-20s \t %-10s \t %-8s \t %s%n", "Name", "State", "Priority", "Type");

System.***out***.println("---------------------------------------------------------");

// Iterate and display information for each thread

**for** (Thread t : threads) {

String name = t.getName();

Thread.State state = t.getState();

**int** priority = t.getPriority();

String type = t.isDaemon() ? "Daemon" : "Normal";

System.***out***.printf("%-20s \t %-10s \t %-8d \t %s%n", name, state, priority, type);

}

t1.start();

System.***out***.println("T1 State :"+t1.getState());

System.***out***.println("T2 State :"+t2.getState());

t2.start();

System.***out***.println("T1 State :"+t1.getState());

System.***out***.println("T2 State :"+t2.getState());

System.***out***.println("Is T1 Alive :" + t1.isAlive());

System.***out***.println("Is T2 Alive :" + t2.isAlive());

System.***out***.println("Is T1 Daemon :" + t1.isDaemon());

System.***out***.println("Is T2 Daemon :" + t2.isDaemon());

System.***out***.println("Is T1 Virtual :" + t1.isVirtual());

System.***out***.println("Is T2 Virtual :" + t2.isVirtual());

t1.*sleep*(1000);

t2.*sleep*(1000);

// t1.stop();

// t2.stop();

System.***out***.println("T1 State after sleep :"+t1.getState());

System.***out***.println("T2 State after sleep:"+t2.getState());

System.***out***.println("Is T1 Alive :" + t1.isAlive());

System.***out***.println("Is T2 Alive :" + t2.isAlive());

}

}

Type of Thread – Deamon (It will be always executed in the background)

**package** com.example;

**public** **class** RunnableDemo {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Runnable r1 = **new** SpellCheckThread();

Runnable r2 = **new** GrammerCheckThread();

Thread t1 = **new** Thread(r1, "Specll-Check");

Thread t2 = **new** Thread(r2, "Grammer-Check");

System.***out***.println("T1 Name:"+t1.getName());

System.***out***.println("T2 Name:"+t2.getName());

System.***out***.println("T1 State:"+t1.getState());

System.***out***.println("T2 State:"+t2.getState());

t1.start();

t2.start();

t2.*sleep*(1000);

System.***out***.println("T1 State:"+t1.getState());

t1.*sleep*(1000);

System.***out***.println("T2 State:"+t2.getState());

}

}

**class** SpellCheckThread **implements** Runnable {

@Override

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

// **TODO** Auto-generated method stub

System.***out***.println("Thread is Running now!!!!");

System.***out***.println(Thread.*currentThread*().getName() + " - Spell Check Thread is running");

}

}

**class** GrammerCheckThread **implements** Runnable {

@Override

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

// **TODO** Auto-generated method stub

System.***out***.println("Thread is Running now!!!!");

System.***out***.println(Thread.*currentThread*().getName() + " - Grammer Check Thread");

}

}

ThreadGroup

**package** com.example;

**public** **class** ThreadGroupDemo {

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

MyThread t1 = **new** MyThread();

MyThread t2 = **new** MyThread();

Runnable r1 = **new** Runnable() {

@Override

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

// **TODO** Auto-generated method stub

System.***out***.println("Anonymous Runnable Class- Run method!!!");

}

};

// starting the thread

t1.start();

t2.start();

r1.run();

// getting the group of the threads/

ThreadGroup threadGroup = Thread.*currentThread*().getThreadGroup();

// getting the total active count of the threads

**int** threadCount = threadGroup.activeCount();

Thread threadList[] = **new** Thread[threadCount];

// enumerating over the thread list

threadGroup.enumerate(threadList);

System.***out***.println("Active threads are:");

// iterating over the for loop to get the names of

// all the active threads.

**for** (**int** i = 0; i < threadCount; i++)

System.***out***.println(threadList[i].getName());

}

}

**class** MyThread **extends** Thread {

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

System.***out***.println(Thread.*currentThread*().getName() + " - is running!!!");

}

}

Single or Multi-player Games

* 1. Display current driver Speed, Name, Position in the race.
  2. Score board