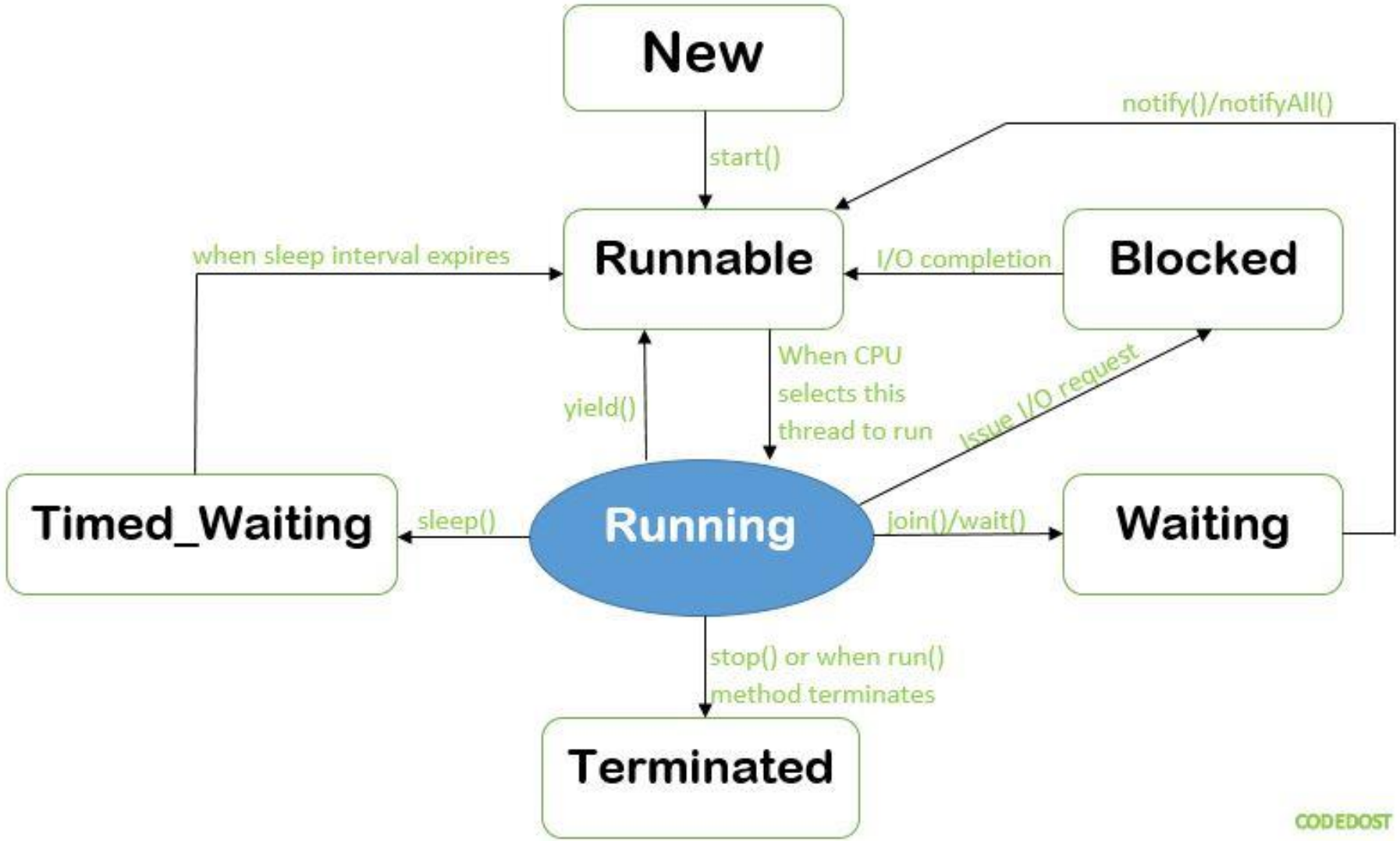
Multithreading-2

Praveen bhosle

Life Cycle of a Thread in JAVA



1. New or Born State-

- The thread is in new state if you create an instance of Thread class but before the invocation of start () method.

obj.

Thread t = new Thread();
t.start();

2. Runnable state-

- The thread is in runnable state after invocation of start () method, but the thread scheduler has not selected it to be the running thread.

3. Running state-

CPU

- The thread is in running state if the thread scheduler has selected it.

4. Dead state-

- A thread is in terminated or dead state when its run () method exits.

- A thread in runnable state may run for some time and then get blocked or may enter the waiting or timed_waiting states.

Blocked

- A thread enters the blocked state waiting for a monitor lock to be released.

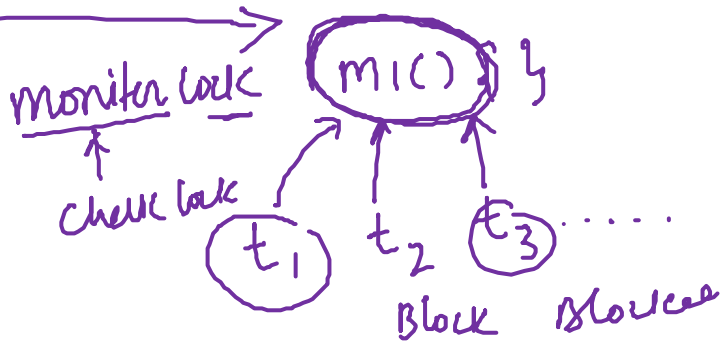
Waiting

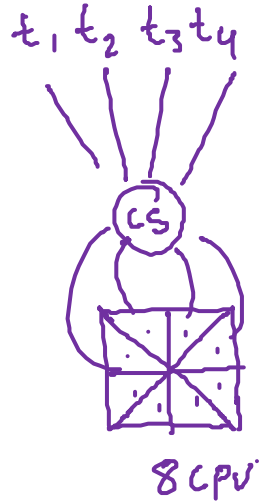
- A thread enters into the waiting state when the thread is waiting for some other thread to perform a particular action without any time limit. For example, calling the join() method, may make a thread enter into the waiting state.

Timed_Waiting

- A thread enters into the timed_waiting state when the thread is waiting for some other thread to perform a specific action for a specified time period. For example calling Thread.sleep(1000) for 1 second, may make a thread enter into timed_waiting state.

Synchronization in JAVA





- Synchronization in java controls multiple threads from accessing the same shared resource in order to prevent an inconsistent state.

- Java Synchronization is done when we want to allow only one thread to access the shared resource.

- For example, let us say A and B are joint account holders of a bank, they both are at different locations and want to withdraw 1000$ each but their account has only 1000$, so now the software will lock one of the transactions and will wait for it to complete, let us say this transaction lock is of A, so as soon as A withdraws 1000$ the balance will be 0, and now B's transaction is executed but will show no balance left, else if it was not synchronized, it would have entered into an inconsistent state.

Understanding it with code….

Synchronized method-

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.
- Example- synchronized void test (){
- // write code here
- }

bank/src/bank/Demo.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Test.java | Demo2.java | Demo.java

```
1  package bank;
2
3  public class Demo implements Runnable {
4
5      @Override
6      public void run() {
7          criticalMethod();
8
9      }
10
11     private synchronized void criticalMethod() {
12
13         String name = Thread.currentThread().getName();
14         System.out.println(name + " has eneterd the critical part of code");
15         for (int i = 1; i <= 8; i++)
16             System.out.println("Name of thread " + name + " and value" + i);
17         System.out.println(name + "has exited the method \n");
18
19     }
20
21 }
22
```

Writable | Smart Insert | 1 : 1 : 0

Type here to search

ENG 21:35 24-08-2022

bank/src/bank/Test.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Test.java | Demo2.java | Demo.java

```
1  package bank;
2
3  public class Test {
4
5      public static void main(String[] args) {
6          Demo demo = new Demo();
7          Thread thread1 = new Thread(demo);
8          Thread thread2 = new Thread(demo);
9          thread1.start();
10         thread2.start();
11     }
12
13 }
14
```

Problems | Javadoc | Console

<terminated> Test (3) [Java Application] C:\Program Files\Java\jdk1.8.0_331\bin\javaw.exe (24-Aug-2022, 9:35:46

```
Name of thread Thread-0 and value8
Thread-0has exited the method

Thread-1 has eneterd the critical part of code
Name of thread Thread-1 and value1
Name of thread Thread-1 and value2
Name of thread Thread-1 and value3
Name of thread Thread-1 and value4
Name of thread Thread-1 and value5
Name of thread Thread-1 and value6
Name of thread Thread-1 and value7
Name of thread Thread-1 and value8
Thread-1has exited the method
```

Type here to search

ENG 21:35 24-08-2022

Synchronized Block-

- Synchronized block can be used to perform synchronization on any specific resource of the method.
- Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

- Note-
- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.
- Syntax-
- synchronized (object reference) {
- //code block
- }

bank/src/bank/Demo2.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Test.java | Demo2.java | Demo.java

```
1  package bank;
2
3  public class Demo2 implements Runnable {
4
5      public void run() {
6
7          int count = 0;
8          synchronized (this) {
9              String name = Thread.currentThread().getName();
10             System.out.println(name + "has eneterd the synchronized block");
11             for (int x = 1; x < 5; x++)
12                 System.out.println("Count =>" + (count++));
13             System.out.println(name + "has exited the block\n");
14
15         }
16
17     }
18
19 }
20
```

Writable | Smart Insert | 8 : 27 [4]

Type here to search

ENG 21:36 24-08-2022

bank/src/bank/Test.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Test.java | Demo2.java | Demo.java

```
1  package bank;
2
3  public class Test {
4
5      public static void main(String[] args) {
6          Demo2 demo = new Demo2();
7          Thread thread1 = new Thread(demo);
8          Thread thread2 = new Thread(demo);
9          thread1.start();
10         thread2.start();
11     }
12
13 }
14
```

Problems | Javadoc | Console

<terminated> Test (3) [Java Application] C:\Program Files\Java\jdk1.8.0_331\bin\javaw.exe (24-Aug-2022, 9:37:05

```
Thread-0has eneterd the synchronized block
Count =>0
Count =>1
Count =>2
Count =>3
Thread-0has exited the block

Thread-1has eneterd the synchronized block
Count =>0
Count =>1
Count =>2
Count =>3
Thread-1has exited the block
```

Type here to search

ENG 21:37 24-08-2022

Interview Question on Multithreading

1>What's the difference between User thread and Daemon thread?

A>User Thread (Non-Daemon Thread): In Java, user threads have a specific life cycle and its life is independent of any other thread. JVM (Java Virtual Machine) waits for any of the user threads to complete its tasks before terminating it. When user threads are finished, JVM terminates the whole program along with associated daemon threads.

Daemon Thread: In Java, daemon threads are basically referred to as a service provider that provides services and support to user threads. There are basically two methods available in thread class for daemon thread: setDaemon() and isDaemon().

What are the wait() and sleep() methods?

- wait(): As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify () or notifyAll() method for the object's monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.
- sleep(): As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn't release the lock while waiting and is mostly used to introduce pause on execution. It is defined in thread class, and no need to call from a synchronized context.

What’s the difference between notify() and notifyAll()?

- notify(): It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object’s monitor.
- notifyAll(): It sends notifications and wakes up all threads and allows them to compete for the object's monitor instead of a single thread.

Why wait(), notify(), and notifyAll() methods are present in Object class?

- We know that every object has a monitor that allows the thread to hold a lock on the object. But the thread class doesn't contain any monitors. Thread usually waits for the object’s monitor (lock) by calling the wait() method on an object, and notify other threads that are waiting for the same lock using notify() or notifyAll() method. Therefore, these three methods are called on objects only and allow all threads to communicate with each that are created on that object.

Runnable Interface	Callable Interface
It does not return any result and therefore, cannot throw a checked exception.	It returns a result and therefore, can throw an exception.
It cannot be passed to invokeAll method.	It can be passed to invokeAll method.
It was introduced in JDK 1.0.	It was introduced in JDK 5.0, so one cannot use it before Java 5.
It simply belongs to Java.lang.	It simply belongs to java.util.concurrent.
It uses the run() method to define a task.	It uses the call() method to define a task.
To use this interface, one needs to override the run() method.	To use this interface, one needs to override the call() method.

What is the start() and run() method of Thread class?

- start(): In simple words, the start() method is used to start or begin the execution of a newly created thread. When the start() method is called, a new thread is created and this newly created thread executes the task that is kept in the run() method. One can call the start() method only once.

- run(): In simple words, the run() method is used to start or begin the execution of the same thread. When the run() method is called, no new thread is created as in the case of the start() method. This method is executed by the current thread. One can call the run() method multiple times.

What’s the purpose of the join() method?

- Join method in Java allows one thread to wait until another thread completes its execution. In simpler words, it means it waits for the other thread to die. It has a void type and throws InterruptedException.

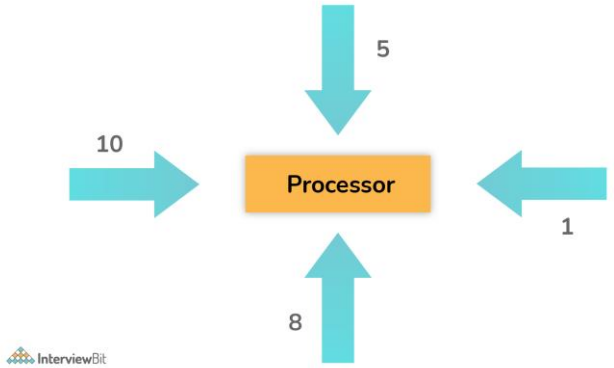
- How do threads communicate with each other?
- Threads can communicate using three methods i.e., wait(), notify(), and notifyAll().

- Can two threads execute two methods ?
- Yes, it is possible. If both the threads acquire locks on different objects, then they can execute concurrently without any problem.

- Can you start a thread twice?
- No, it's not at all possible to restart a thread once a thread gets started and completes its execution. Thread only runs once and if you try to run it for a second time, then it will throw a runtime exception i.e., java.lang.IllegalThreadStateException.
- What is Thread Scheduler?
- Thread Scheduler: It is a component of JVM that is used to decide which thread will execute next if multiple threads are waiting to get the chance of execution. By looking at the priority assigned to each thread that is READY, the thread scheduler selects the next run to execute. To schedule the threads, it mainly uses two mechanisms: Preemptive Scheduling and Time slicing scheduling.

Explain thread priority.

- Thread priority simply means that threads with the highest priority will get a chance for execution prior to low-priority threads. One can specify the priority but it's not necessary that the highest priority thread will get executed before the lower-priority thread. Thread scheduler assigns processor to thread on the basis of thread priority. The range of priority changes between 1-10 from lowest priority to highest priority.



- What will happen if we don’t override the thread class run() method?
- Nothing will happen as such if we don’t override the run() method. The compiler will not show any error. It will execute the run() method of thread class and we will just don’t get any output because the run() method is with an empty implementation.
- Is it possible to call the run() method directly to start a new thread?
- No, it's not possible at all. You need to call the start method to create a new thread otherwise run method won't create a new thread. Instead, it will execute in the current thread.

- Can we call the run() method instead of start()?
- Yes, calling run() method directly is valid, but it will not work as a thread instead it will work as a normal object. There will not be context-switching between the threads. When we call the start() method, it internally calls the run() method, which creates a new stack for a thread while directly calling the run() will not create a new stack.

- What is static synchronization?
- If you make any static method as synchronized, the lock will be on the class not on the object. If we use the synchronized keyword before a method so it will lock the object (one thread can access an object at a time) but if we use static synchronized so it will lock a class (one thread can access a class at a time).

- What is the deadlock?
- Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.
- How to detect a deadlock condition? How can it be avoided?
- We can detect the deadlock condition by running the code on cmd and collecting the Thread Dump, and if any deadlock is present in the code, then a message will appear on cmd.

- Ways to avoid the deadlock condition in Java:

- Avoid Nested lock: Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.
- Avoid unnecessary locks: we must avoid the locks which are not required.
- Using thread join: Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of join method.

- How is the safety of a thread achieved?
- If a method or class object can be used by multiple threads at a time without any race condition, then the class is thread-safe. Thread safety is used to make a program safe to use in multithreaded programming. It can be achieved by the following ways:

- Synchronization
- Using Volatile keyword
- Using a lock based mechanism
- Use of atomic wrapper classes

- What is the volatile keyword in java?
- Volatile keyword is used in multithreaded programming to achieve the thread safety, as a change in one volatile variable is visible to all other threads so one variable can be used by one thread at a time.

- What do you understand by thread pool?
- Java Thread pool represents a group of worker threads, which are waiting for the task to be allocated.
- Threads in the thread pool are supervised by the service provider which pulls one thread from the pool and assign a job to it.
- After completion of the given task, thread again came to the thread pool.
- The size of the thread pool depends on the total number of threads kept at reserve for execution.
- The advantages of the thread pool are :

- Using a thread pool, performance can be enhanced.
- Using a thread pool, better system stability can occur.

- Thank you