What is a thread ?

A thread refers to the smallest unit of execution within a process. It is an independent path of execution that can run concurrently with other threads in a program.

Java programs can have multiple threads, and each thread represents a separate flow of control.

How many way we create a thread

1> Extending the Thread class

2> Implementing Runnable Interface

***3>Implementing Callable Interface

Create a thread using runnable interface

Create a class that implements the Runnable interface. Implement the run method,

which contains the code to be executed by the thread.

Create an instance of the class implementing Runnable.

Create an instance of the Thread class, passing the Runnable instance to its constructor.

Call the start method on the Thread instance to begin the execution of the thread.

Adv : we can achive inheritance benfit

Thread life cycle

New (or Born):

The thread is in this state when an instance of the Thread class is created(Thread t = new Thread), but the start() method has not been called yet.

The thread is not alive at this point.

Runnable:

After calling the start() method, the thread moves to the runnable state.

In this state, the thread is ready to run, and the Java Virtual Machine (JVM) and Thread scheduler can choose it to be the next thread to run.

Running State:

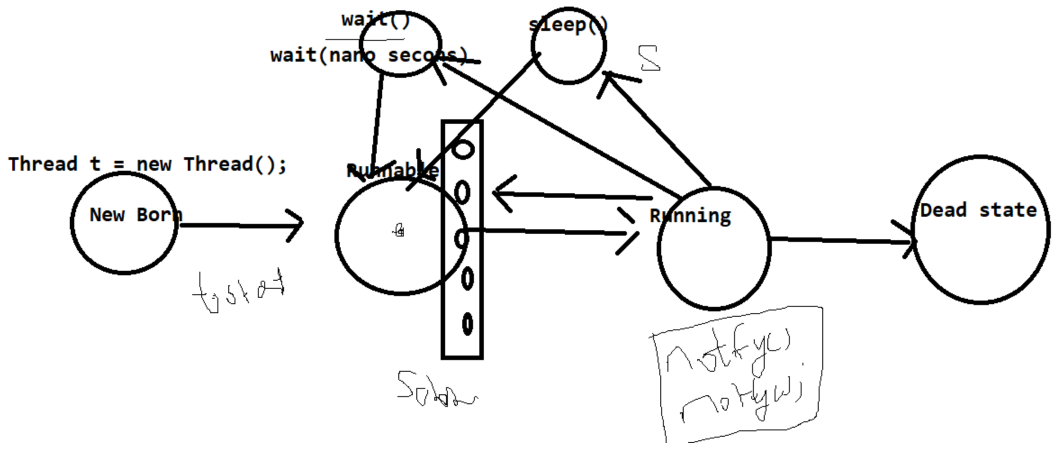
If thread is running state either it completes the task and goes to dead/terminated state.

If task has been not complted moves back to Runnable state.

Terminated (or Dead):

A thread enters the terminated state when its run() method completes

Once a thread is terminated, it cannot be started again.



Thread.yeild--static

The Thread.yield() method in Java is a static method that is used to hint to the scheduler that the current thread is willing to yield its current use of the CPU.

It's a way for the current thread to voluntarily pause and allow other threads of the same priority to run.

join : --throws InterruptedException return type void

join() method is a method provided by the Thread class that allows one thread to wait for the completion of another thread.

***When a thread invokes the join() method on another thread, it essentially waits for that thread to finish its execution before continuing.

Sleep: throws InterruptedException

Thread.sleep() method is used to pause the execution of the current thread for a specified amount of time.

Interupt

If you call interupt method on any thread, If that particular thread in sleep state, It will riase interupted

Exception.

if threas is not in sleep state even though calling interupt won't be any Interupted exception.

Create a   thread using Callable Interface

Thread pool

Future Object and Get methods

The Callable interface is a part of the java.util.concurrent package ,It is similar to the Runnable interface, but It contains call() method having the return type and throw checked exceptions.

We create an thread by callingExecutorService.

We submit the Callable task to the executor using the submit method,

which returns a Future object representing the result of the call method.

While the Callable is running, the main thread can perform other tasks.

We use the get() method on the Future object to retrieve the result of the call method. This method blocks until the result is available.

Finally, we shut down the ExecutorService when we are done.

Deadlock:

Two threads waiting for each other to release the lock.

Below is the program.

```
public class DeadlockExample {

    public static void main(String[] args) {

        final Object resource1 = new Object();

        final Object resource2 = new Object();


        // Thread 1

        Thread thread1 = new Thread(() -> {

            synchronized (resource1) {
                System.out.println("Thread 1: Acquired lock on resource1");

                try {

                    // Introducing a delay to increase the chance of deadlock

                    Thread.sleep(100);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }


                System.out.println("Thread 1: Waiting for lock on resource2");

                synchronized (resource2) {

                    System.out.println("Thread 1: Acquired lock on resource2");

                }

            }

        });


        // Thread 2

        Thread thread2 = new Thread(() -> {

            synchronized (resource2) {

                System.out.println("Thread 2: Acquired lock on resource2");

                try {

                    Thread.sleep(100);

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }
```

```
                    System.out.println("Thread 2: Waiting for lock on resource1");
                    synchronized (resource1) {
                        System.out.println("Thread 2: Acquired lock on resource1");
                    }
                }
            });

            // Start the threads
            thread1.start();
            thread2.start();
        }
    }
```

Deamon thread :

A daemon thread is a thread that runs in the background to provide the support to child threads.

User thread can be set to deamon by calling setDeamon(true).

Since the daemon thread is not essential for the application, it will be terminated when the main thread exits.

Background Execution:

Terminated on Program Exit:

If all non-daemon threads have finished their execution, the JVM will exit, and any remaining daemon threads will be abruptly terminated.

Marked with setDaemon():

You can set a thread as a daemon by calling the setDaemon(true) method before starting it. The default value is false, meaning the thread is a user thread.

Synchronization

Synchronization is applied to methods.

If we are calling any instance method as Syn.

It will obtain Object level lock and allows only one thread at any point of time.

Object level lock

Thread acquire the Object lock and after the completion of thread task it will release the lock.

Mean while if any thread try to call on same method, It will wait until lock get released, so thread remains in waiting state.

*** we can provide data conssitency by applying the method syn, but the problem in the Object level lock is that, If the class contains mutiple instance. we cannot achieve the data consistency

Reason: when one thread acting one instance acquiring the lock, where other thread can act on other instance to acquire the lock.

Class level lock

If you mentioned syn for static method, it will create the Class level lock.

Even though threads coming from multiple instance it has to wait for thread to release the lock.

synchronized   Block

Inspite of applying complete method as syn, I would like to apply for particular piece of code than we need to call syn()block

1> syn(this)block with this keyword. It act like a Object level lock (It fails to provide the data consistenct in the case of mutiple instnace)

2>syn(classname.class)block . It act like a Class level lock

//setPriority(1 to 10)

one is leas priority and 10 is the highest priority

Priority value set from 1 to 10 no issues

in case anything greater than 10.(Illegal argument exception)

//IllegalThreadStateException

If thread is allready created on that particular instance, If you try to call again start method from the same instance, It leads to Illegal thread state Exception

What happened if you Overide start methods.

Our class does'nt not have any logic for thread creation, It could be like normal method call.

Thread won't be created.

What happened if you not Overide Run methods.

Created thread won't come to our class. It goes thread class run method.

Reason: we overide the run method and extends the thread class and creates the subclass instance to call a start method, It will automatically creates a thread,

Since it has been called by subclass instance. Creted thread comes to sub class run method. Instead of thread class run method.

Disadvantages of thread creation by Extending the Thread class.

we are missing the inhertianc benfit.

to overcome this problem we creat a thread using Runnable Interface.

wait

notify

notify all

Inter Thread communication --Bank Program