30: [Thread Creation, Thread Lifecycle and Inter-Thread Communication] | Part 2

Multi Threading Part 2

We will cover

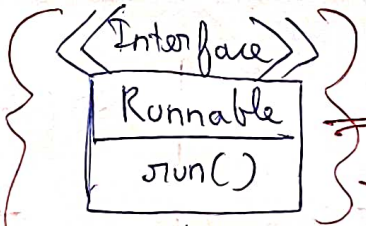
1) Creating Threads
2) Thread lifecycle
3) Thread Synchronisation and Thread Safety
   * Monitor Locks
   * Synchronised blocks
   * Synchronised methods

Ways of Creating a Thread.

1. Implements Runnable Interface ⇒ Used in Industry, we will see soon why.
2. extending Thread Class.

If the class wants it can extend some other class since it can implement multiple

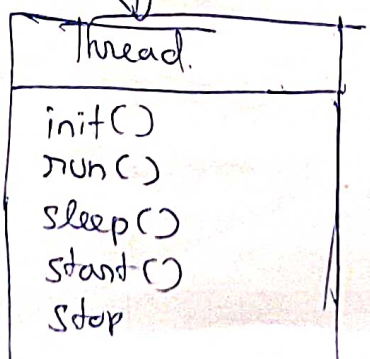
Runnable Interface.



functional interface

Not a thread. This is a normal functional Interface.

implements



This is actually the thread class which takes care of Life Cycle of a thread.

```
class MyRunnableThread implements Runnable {

    @Override
    public void run(){
        sout("implementing Run method");
    }
}.
```

This is a normal class only

But.

Placing this Object into the thread's constructor give it thread's powers

```
Inside main Method{

    Main(){

        MyRunnableThread runnableObj = new MyRunnableThread();
        Thread t = new Thread(runnableObj);

        t.start(); => thread has been starts.
    }.
```

will internally call t.run() run method)

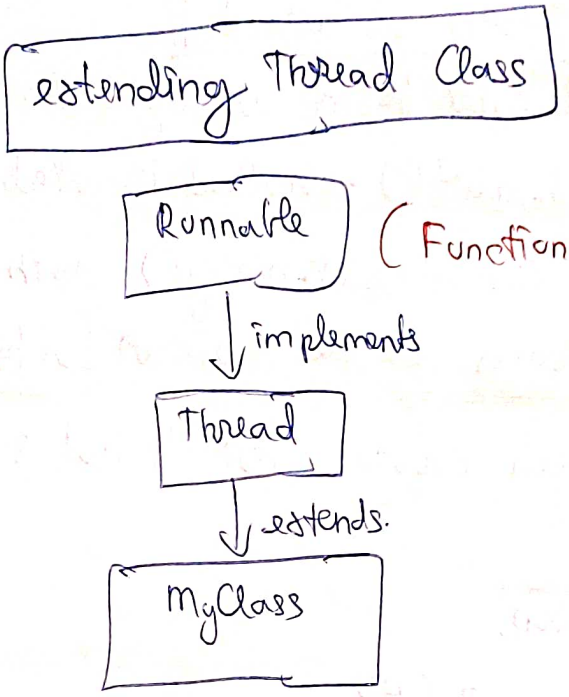
```
Thread - class {

    Implementation of run method inside Thread Class

    @Override
    public void run(){
        if(target != null){
            target.run();
        }
    }.
```

```
    private Runnable target.
```

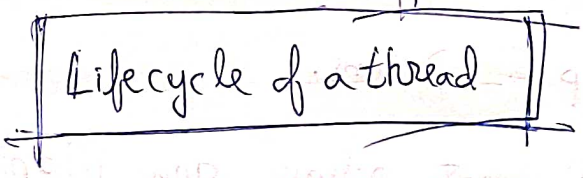
This constructor is responsible for setting the Runnable Object.

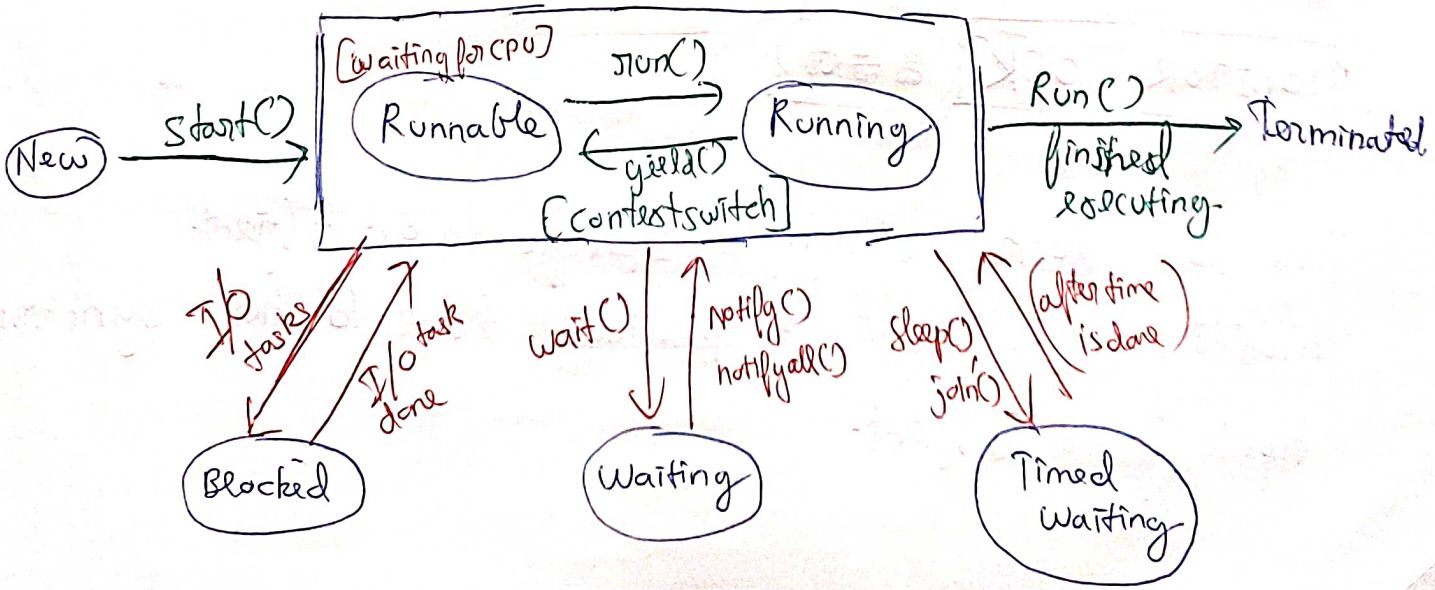


(Step 1): Extend the Thread Class

```
class MyThread extends Thread{
    @Override → [Because target is empty]
    public void run(){
        sout("Run the class");
    }
}.
```

```
Main()
    MyThread t1 = new MyThread();
    t1.start();
```





States of Thread

① Waiting State → The thread will be in waiting till.
When thread.wait() → method is called. It will keep on waiting till we call the thread.notify() method.
(***) [Releases all the monitor locks]

② New → Thread has been created but its not started yet

③ Runnable → Ready to Run,
But waiting for CPU

④ Running → thread is executing code.

⑤ Blocked → ① I/O read is going on.
② Waiting for Lock on an Object of synchronized block.
③ Releases all Monitor Locks.

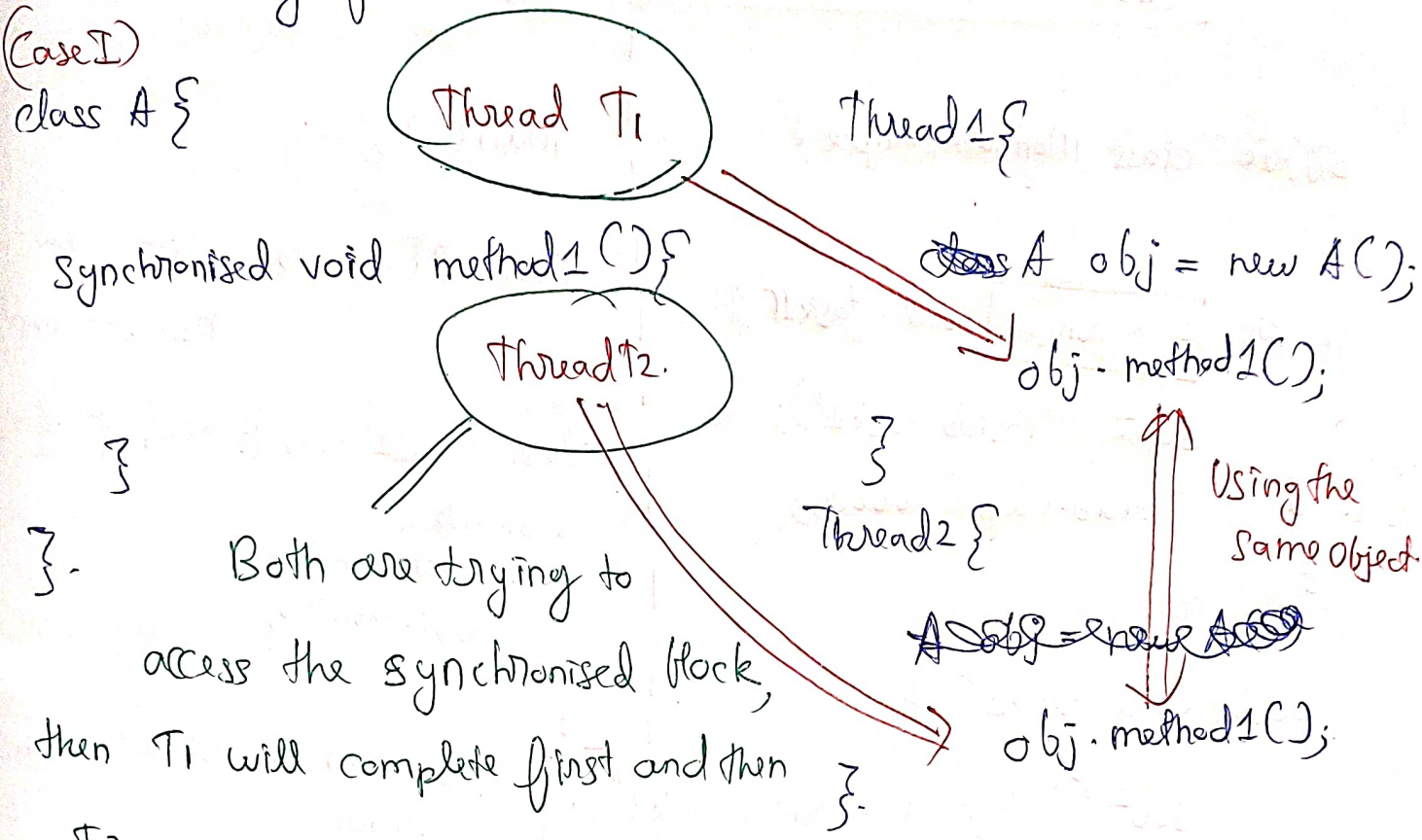
⑥ Timed Waiting →
Sleep ( ) in ms.
*** Do not release any Monitor Lock.

MONITOR LOCK (***)

This is a lock which is specific to an object.
Monitor Lock is meant to provided lock to the synchronized block of the object.

Working of Monitor Locks.

(Case I)



(Case 2)

```
A obj1 = new A();
A obj2 = new A();

Thread 1 {
        obj1.method1();
}.
Thread 2 {
    obj2.method1();
}.
```

Since both are different objects obj1 and obj2. Hence thread1 and thread2 will execute parallely together.

Code Example with Monitor Locks.

```
public class MonitorExample {

    public synchronized void task1() {
        Sout("inside task1");
        Thread.sleep(10000);
    }

    public void task2() {
        Sout("before synchronised task2");
        synchronised (this) {
            Sout("task2 inside synchronised");
        }
    }

    public void task3() {
        Sout("task3");
    }
}

Thread1 t1 = new Thread1();
Thread2 t2 = new Thread2();
Thread3 t3 = new Thread3();
t1.start();
t2.start();
t3.start();
```

Sync Block

```
main() {
    MonitorExample obj = new MonitorExample();

    class Thread1 extends Thread {
        @override
        public void run() {
            obj.task1();
        }
    }

    class Thread2 extends Thread {
        @override
        public void run() {
            obj.task2();
        }
    }

    class Thread3 extends Thread {
        @override
        public void run() {
            obj.task3();
        }
    }
```

Output

```
inside task1
before synchronised task2
task3
task2 inside synchronised.
```

(88) From the output see, how.

task1() → acquired the monitor lock for 10 seconds because it had the sleep().

(88) When Thread2 tried to execute task2, it was able to enter the non-sync part, but had to wait for Thread1 to complete to give back the Monitor Lock to Thread2.

(88) Thread3 had no issues executing as it was a non-sync block.

[Implement Producer, Consumer Problem Yourself]