Agenda

- Multitasking
- Thread Scheduler
- How to Create a Thread
- How to Create a Thread
- Thread Lifecycle
- Thread Name
- Synchronization
- difference

Multitasking

Executing several task simultaneously is called “Multi Tasking ”

There are two types of multitasking :

1. Processed Based Multitasking
2. Thread Based Multitasking

1. Processed Based Multitasking

Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called process based multitasking

2. Thread Based Multitasking

- **Executing several tasks simultaneously where each task is a separate independent part of the same program, is called Thread based multitasking. And each independent part is called a "Thread".**

Thread Scheduler

- **If multiple Threads are waiting to execute then which Thread will execute 1st is decided by "Thread Scheduler" which is part of JVM.**

- **Which algorithm or behaviour followed by Thread Scheduler we can't expect exactly it is the JVM vendor dependent hence in multithreading examples we can't expect exact execution order and exact output.**

How to Create a Thread

By extending Thread class.

By implementing Runnable interface.

How to Create a Thread

By extending Thread class.

MyThreadMain.java

```java
package com.simplilearn.thread;

public class MyThreadMain {

    public static void main(String[] args) {

        // 1. Instantiation of a Thread Class
        MyThread myThread = new MyThread();
        // 2. starting of a Thread
        myThread.start();

    }

}
```

MyThread.java

```java
package com.simplilearn.thread;

public class MyThread extends Thread {

    @Override
    public void run() {
        super.run();

        for (int i = 0; i < 5; i++) {

            System.out.println("Thread: From MyThread Class");
        }

    }
}
```

How to Create a Thread

By implementing Runnable interface.

MyThread.java

```java
package com.simplilearn.thread;

public class MyThread implements Runnable {

    @Override
    public void run() {

        for (int i = 0; i < 5; i++) {

            System.out.println("Thread: From MyThread Class");
        }

    }
}

```

MyThreadMain.java

```java
package com.simplilearn.thread;

public class MyThreadMain {

    public static void main(String[] args) {

        // 1. Instantiation of a user defined Thread Class
        MyThread myThread = new MyThread();

        // 2. Instantiation of a Thread Class targeting the user defined Thread Class object
        Thread thread = new Thread(myThread);

        thread.start();
    }
}
```

What does t.start() method do ?

For every Thread the required mandatory activities like registering the Thread with Thread Scheduler will takes care by Thread class start() method and programmer is responsible just to define the job of the Thread inside run() method.

Register Thread with Thread Scheduler

All other mandatory low level activities

Invoke or calling run() method

Difference between t.start() and t.run() methods.

- In the case of t.start() a new Thread will be created which is responsible for the execution of run() method.

- But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread.

MyThreadMain.java

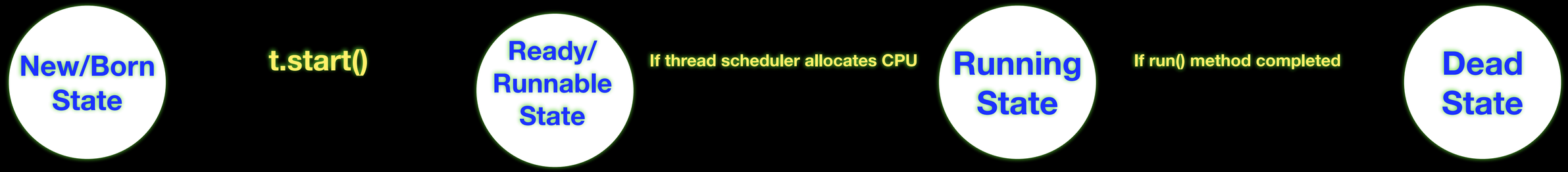
```java
package com.simplilearn.thread;

public class MyThreadMain {

    public static void main(String[] args) {

        // 1. Instantiation of a Thread Class
        MyThread myThread = new MyThread();
        // 2. starting of a Thread
        myThread.start();
        // 3. Calling a run() method from MyThread Class not Thread Class
        myThread.run();

    }

}
```

MyThread.java

```java
package com.simplilearn.thread;

public class MyThread extends Thread {

    @Override
    public void run() {
        super.run();

        for (int i = 0; i < 5; i++) {

            System.out.println("Thread: From MyThread Class");
        }

    }
}

```

Thread Lifecycle

- Once we created a Thread object then the Thread is said to be in new state or born state.
- Once we call start() method then the Thread will be entered into Ready or Runnable state.
- If Thread Scheduler allocates CPU then the Thread will be entered into running state.
- Once run() method completes then the Thread will entered into dead state.



- After starting a Thread we are not allowed to restart the same Thread once again otherwise we will get runtime exception saying "IllegalThreadStateException"

Thread Name

- Every Thread in java has some name it may be provided explicitly by the programmer or automatically generated by JVM

- Thread class defines the following methods to get and set name of a Thread.

- Methods :

```
public final String getName()
```

```
public final void setName(String name)
```

Thread Priorities

Every Thread in java has some priority it may be default priority generated by JVM (or) explicitly provided by the programmer.

The valid range of Thread priorities is 1 to 10[but not 0 to 10] where 1 is the least priority and 10 is highest priority

Thread class defines the following constants to represent some standard priorities

Thread. MIN_PRIORITY ———— 1

Thread. MAX_PRIORITY ———— 10

Thread. NORM_PRIORITY ———— 5

- Thread scheduler uses these priorities while allocating CPU.
- The Thread which is having highest priority will get chance for 1st execution
- If 2 Threads having the same priority then we can't expect exact execution order it depends on Thread scheduler whose behaviour is vendor dependent

Thread Priorities

Every Thread in java has some priority it may be default priority generated by JVM (or) explicitly provided by the programmer.

The valid range of Thread priorities is 1 to 10[but not 0 to 10] where 1 is the least priority and 10 is highest priority

- We can get and set the priority of a Thread by using the following methods.

```
public final int getPriority();
```

```
public final int setPriority(int newPriority);
```

- Applicable values for newPriority is 1 to 10 , except this value will throw runtime exception saying "IllegalArgumentException".

Default Thread Priorities

The default priority only for the main Thread is 5.

For the inheriting thread , the default priority is depends up on Parent class thread priorities.

Prevent the Thread Execution

yield();

join();

sleep();

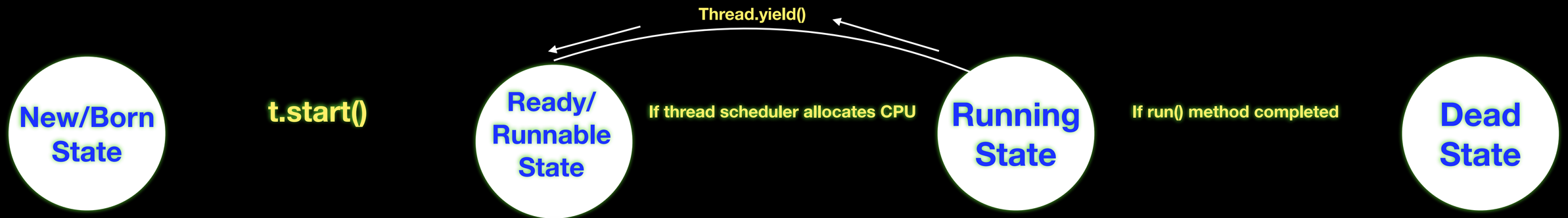
public static native void yield(); Method in Thread

yield() method causes "to pause current executing Thread for giving the chance of remaining waiting Threads of same priority"

If all waiting Threads have the low priority or if there is no waiting Threads then the same Thread will be continued its execution

If several waiting Threads with same priority available then we can't expect exact which Thread will get chance for execution

The Thread which is yielded when it get chance once again for execution is depends on mercy of the Thread scheduler.



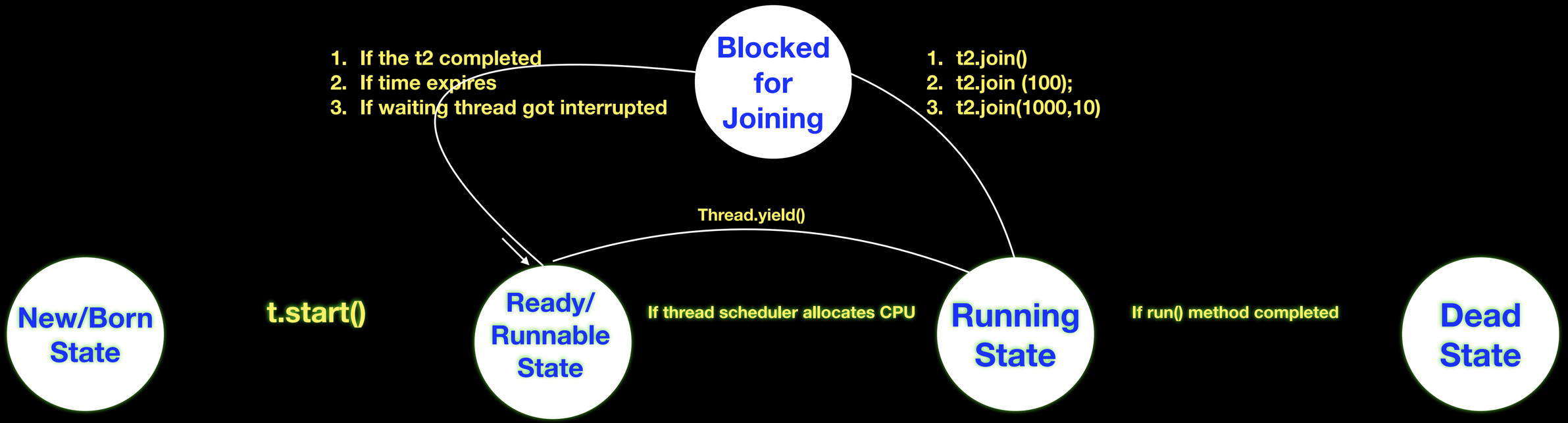
public final void join() ; Method in Thread

If a Thread wants to wait until completing some other Thread then we should go for join() method.

public final void join()throws InterruptedException

public final void join(long ms) throws InterruptedException

public final void join(long ms,int ns) throws InterruptedException

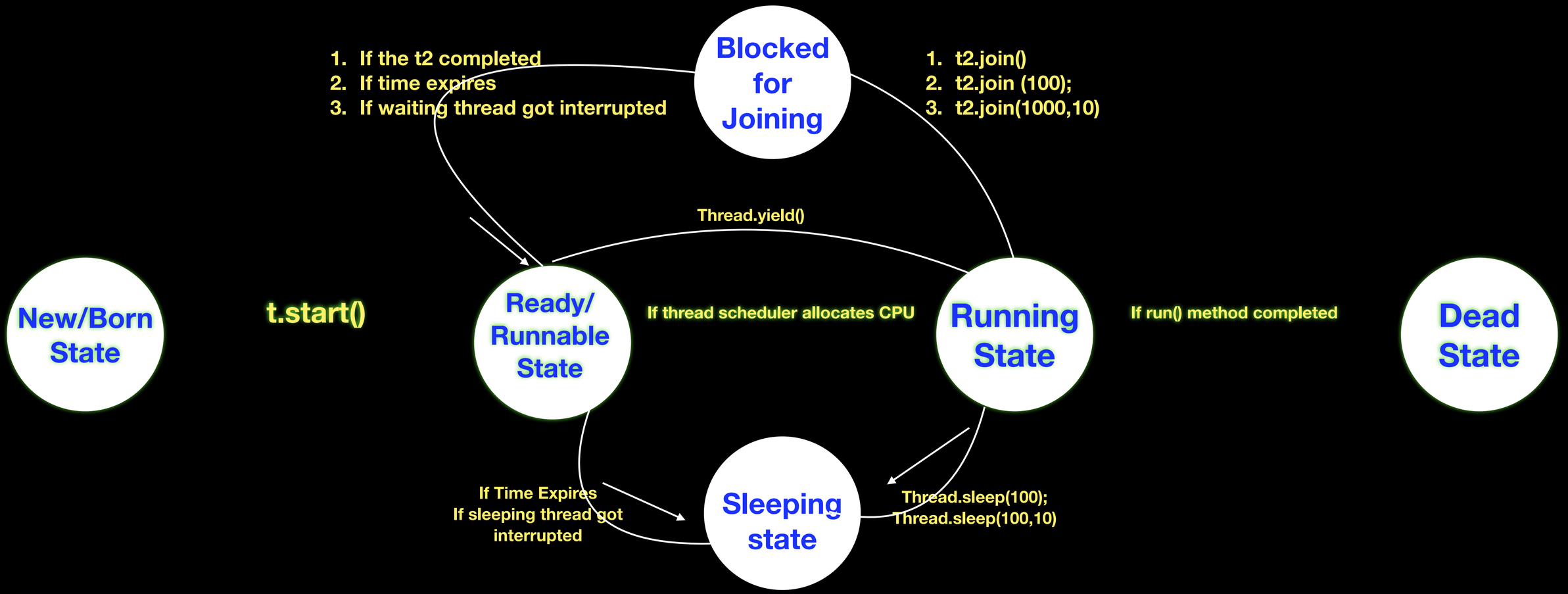


public static void sleep(…) ; Method in Thread

If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.

public static void sleep(long mls) throws InterruptedException

public static void sleep(long mls, int n) throws InterruptedException



Inter Thread Communication

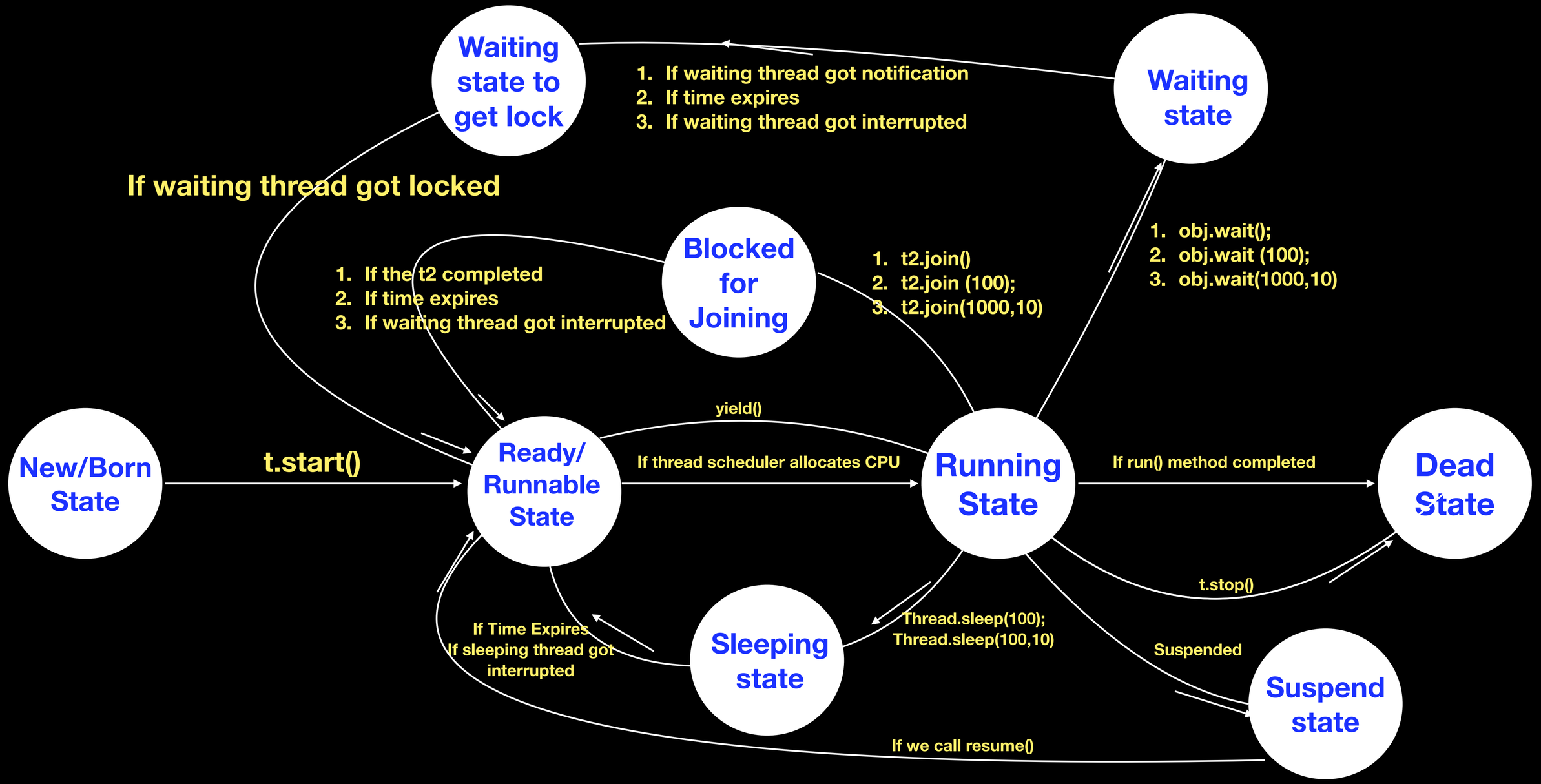
wait();

notify();

notifyAll();

- **Two Threads can communicate with each other by using wait(), notify() and notifyAll() methods.**
- **wait(), notify() and notifyAll() methods are available in Object class but not in Thread class because Thread can call these methods on any common object.**
- **To call wait(), notify() and notifyAll() methods compulsory the current Thread should be owner of that object**
- **current Thread should be in synchronized area. Hence we can call wait(), notify() and notifyAll() methods only from synchronized area otherwise we will get runtime exception saying IllegalMonitorStateException.**
- **Once a Thread calls wait() method on the given object 1st it releases the lock of that object immediately and entered into waiting state.**
- **Once a Thread calls notify() (or) notifyAll() methods it releases the lock of that object but may not immediately.**
- **Except these (wait(),notify(),notifyAll()) methods there is no other place(method) where the lock release will be happen.**

Thread Kill or Interruption


Waiting state to get lock
1. If waiting thread got notification
2. If time expires
3. If waiting thread got interrupted
Waiting state
If waiting thread got locked
Blocked for Joining
1. If the t2 completed
2. If time expires
3. If waiting thread got interrupted
1. t2.join()
2. t2.join (100);
3. t2.join(1000,10)
1. obj.wait();
2. obj.wait (100);
3. obj.wait(1000,10)
yield()
New/Born State
t.start()
Ready/ Runnable State
If thread scheduler allocates CPU
Running State
If run() method completed
Dead State
t.stop()
If Time Expires
If sleeping thread got interrupted
Sleeping state
Thread.sleep(100);
Thread.sleep(100,10)
Suspended
Suspend state
If we call resume()

Synchronization

Synchronized is the keyword applicable for methods and blocks but not for classes and variables.

If a method or block declared as the synchronized then at a time only one Thread is allow to execute that method or block on the given object.

The main advantage of synchronized keyword is we can resolve date inconsistency problems.

But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system

Hence if there is no specific requirement then never recommended to use synchronized keyword.

Internally synchronization concept is implemented by using lock concept.

If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object. Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object. If the synchronized method execution completes then automatically Thread releases lock

While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method simultaneously. [lock concept is implemented based on object but not based on method]

Synchronization Block

If very few lines of the code required synchronization then it's never recommended to declare entire method as synchronized we have to enclose those few lines of the code with in synchronized block

The main advantage of synchronized block over synchronized method is it reduces waiting time of Thread and improves performance of the system

Deamon Thread

The Threads which are executing in the background are called daemon Threads.The main objective of daemon Threads is to provide support for non-daemon Threads like main Thread.

Difference Between yield() , join() , sleep()

Property	yield()	join()	sleep()
Purpose	To pause current executing thread for giving the chance of remaining waiting thread of same priority .	If a thread wants to wait until completing some other Thread then we should got for join.	If a Thread do not want to perform any operation for a particular amount of time then we should go for sleep().
Is it static ?	YES	NO	YES
Is it final ?	NO	YES	NO
Is it Overloaded ?	NO	YES	YES
Is it throws ?	NO	YES	YES
Is it native method?	YES	NO	**sleep(long ms) -- >native sleep(long ms,int ns) -->non-native**

Difference Between wait() , sleep()

wait()	sleep()
This method belongs to Object Class	This method belongs to Thread Class
wait() method release the thread lock during Synchronisation	sleep() method does not release the thread lock during Synchronisation
This method only can be called from Synchronisation context	Synchronisation context not required to call sleep() method.

Thank You
Happy Learning
Keep Watching