INTRODUCTION TO THREAD

Silicon University

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University

Introduction

- A thread is a single flow of control like simple program.
- A unique property of java is multithreading only because java supports multithreading.
- More than one thread (program) run simultaneously is known as multithreading (multiprogramming).
- In multithreading java interpreter handles the switching of control between the threads in such a way that it looks like they are running concurrently.
- Multithreading is useful in a number of ways. We can divide a long program into number of threads and executes them in parallel.

The Main Thread

- When our simple program starts one single thread begins running immediately.
- This is called our single main thread.
- The main thread create automatically when program is started.
- It is very important thread because of two reason.

1.) From the main thread other child thread will be created.

2.) Main thread is all most every time stop running lastly because it has to remove or shutdown few resources as well as few action.

- Actually by calling the method currentThread() of Thread class we can control our main thread.

Example

```
public class Main_Thread
{
  public static void main(String args[])
  {
    Thread t = Thread.currentThread();
    System.out.println("Current thread: " + t);

    // change the name of the thread
    t.setName("My Thread");
    System.out.println("After name change: " + t);
    try
    {
```

```
 for(int n = 5; n > 0; n--)
            {
               System.out.println(n);  //print number with interval of 
1 sec.
              Thread.sleep(1000);     //Thread is going to sleep for 1 
sec.
            }
        }
        catch (InterruptedException e)
        {
           System.out.println("Main thread interrupted");
        }
    }
}
```

Output:

```
Current thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
5
4
3
2
1
```

Here first of all we give reference of our current main single thread to t by thread object and currentThread() method.

The number 5 to 1 will be print at interval of 1 second due to sleep method.

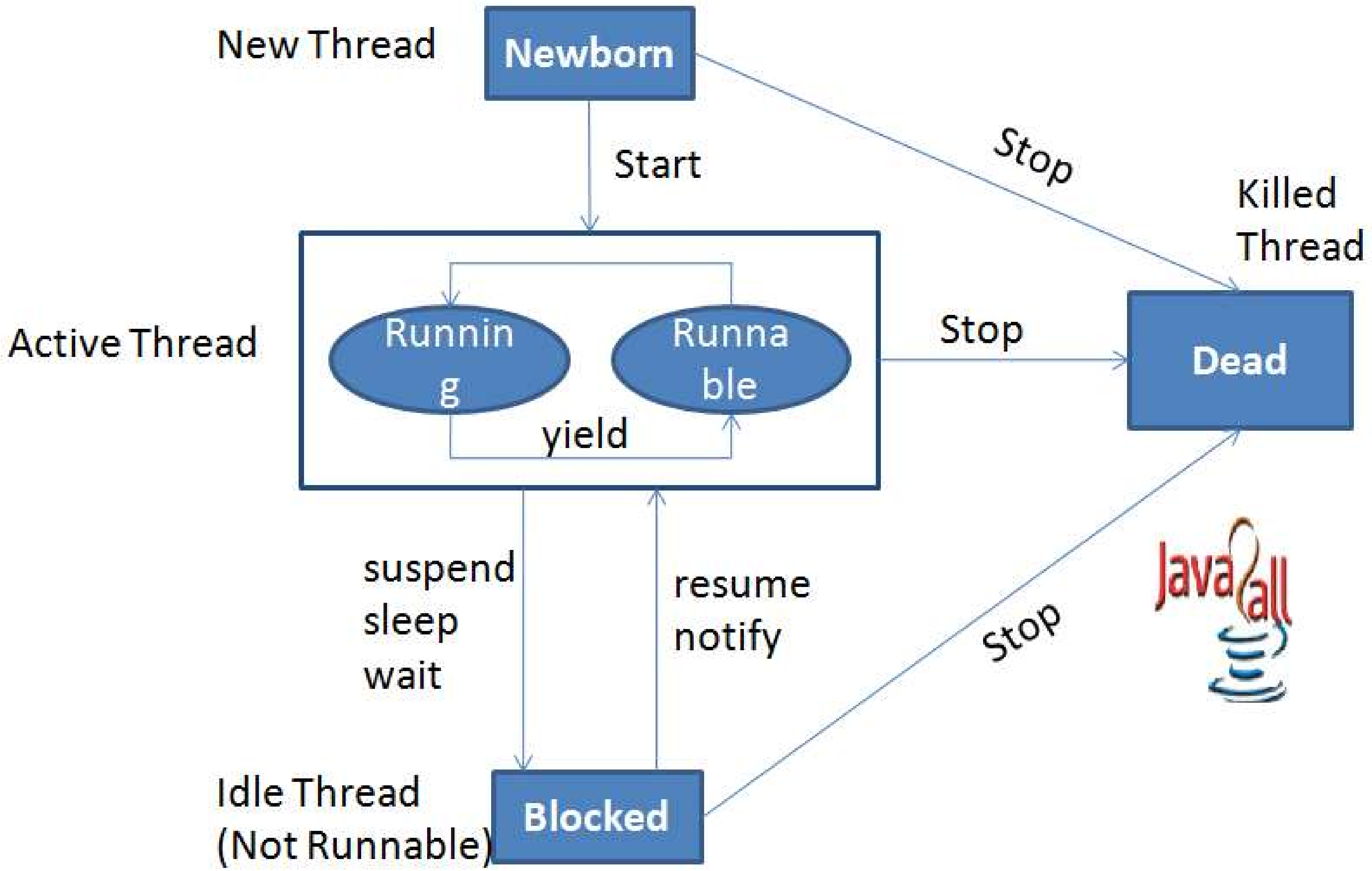
Thread will go to sleep for 1000 ms. due to sleep method

Thread Life Cycle:

Thread has many different state through out its life.

- Newborn State
- Runnable State
- Running State
- Blocked State
- Dead State

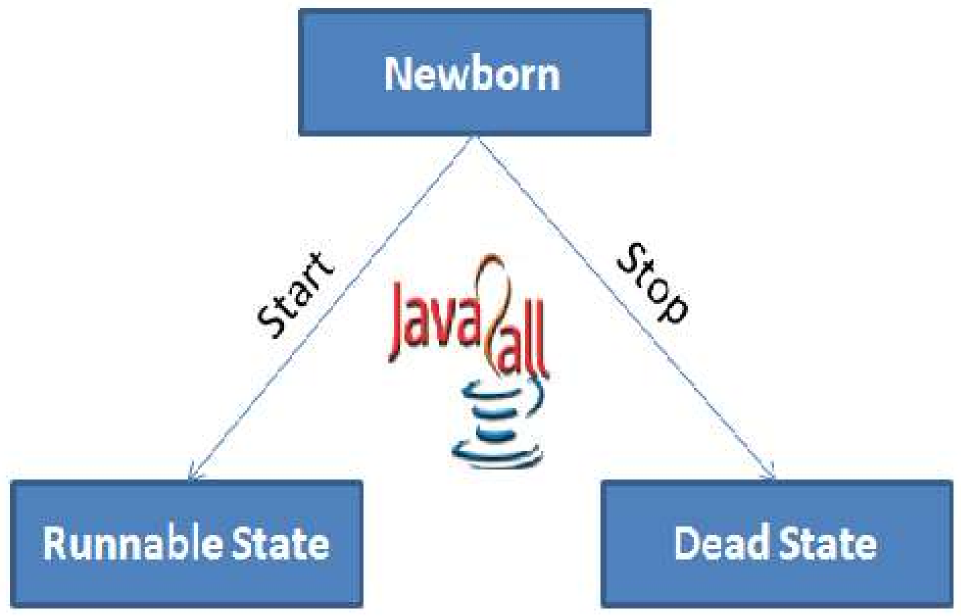
Thread should be in any one state of above and it can be move from one state to another by different methods and ways.



State transition diagram of a thread

Newborn State

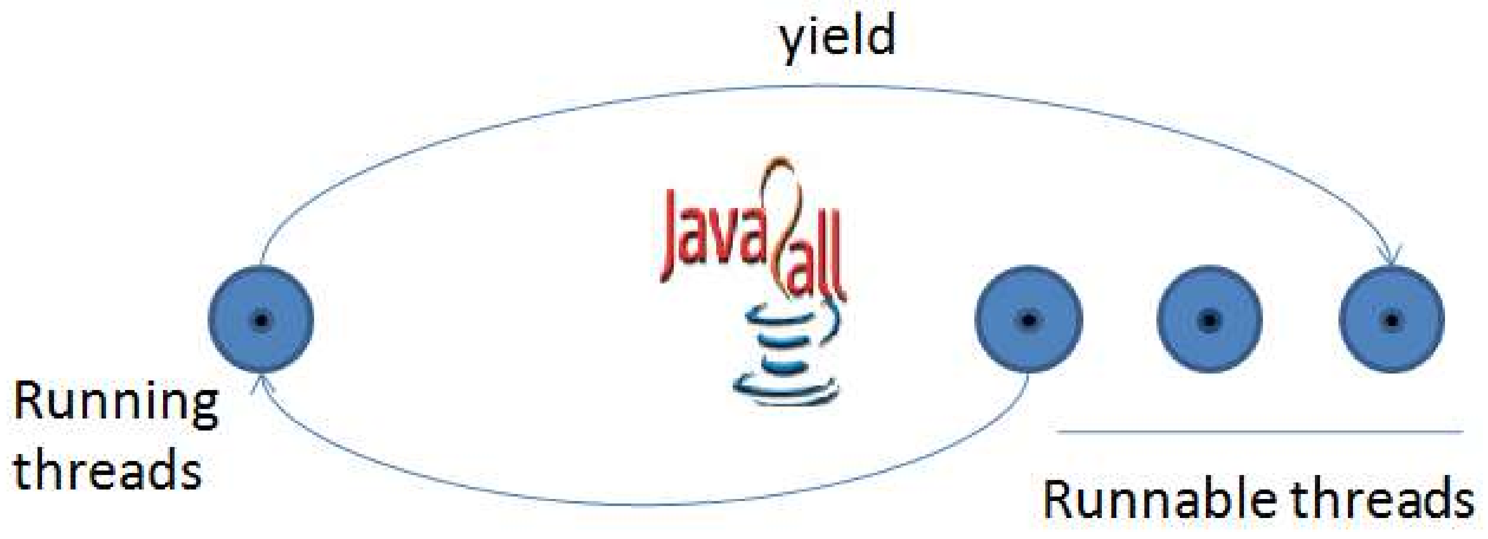
- When we create a thread it will be in Newborn State.
- The thread is just created still its not running.
- We can move it to running mode by invoking the start() method and it can be killed by using stop() method.



Scheduling a newborn thread

Runnable State

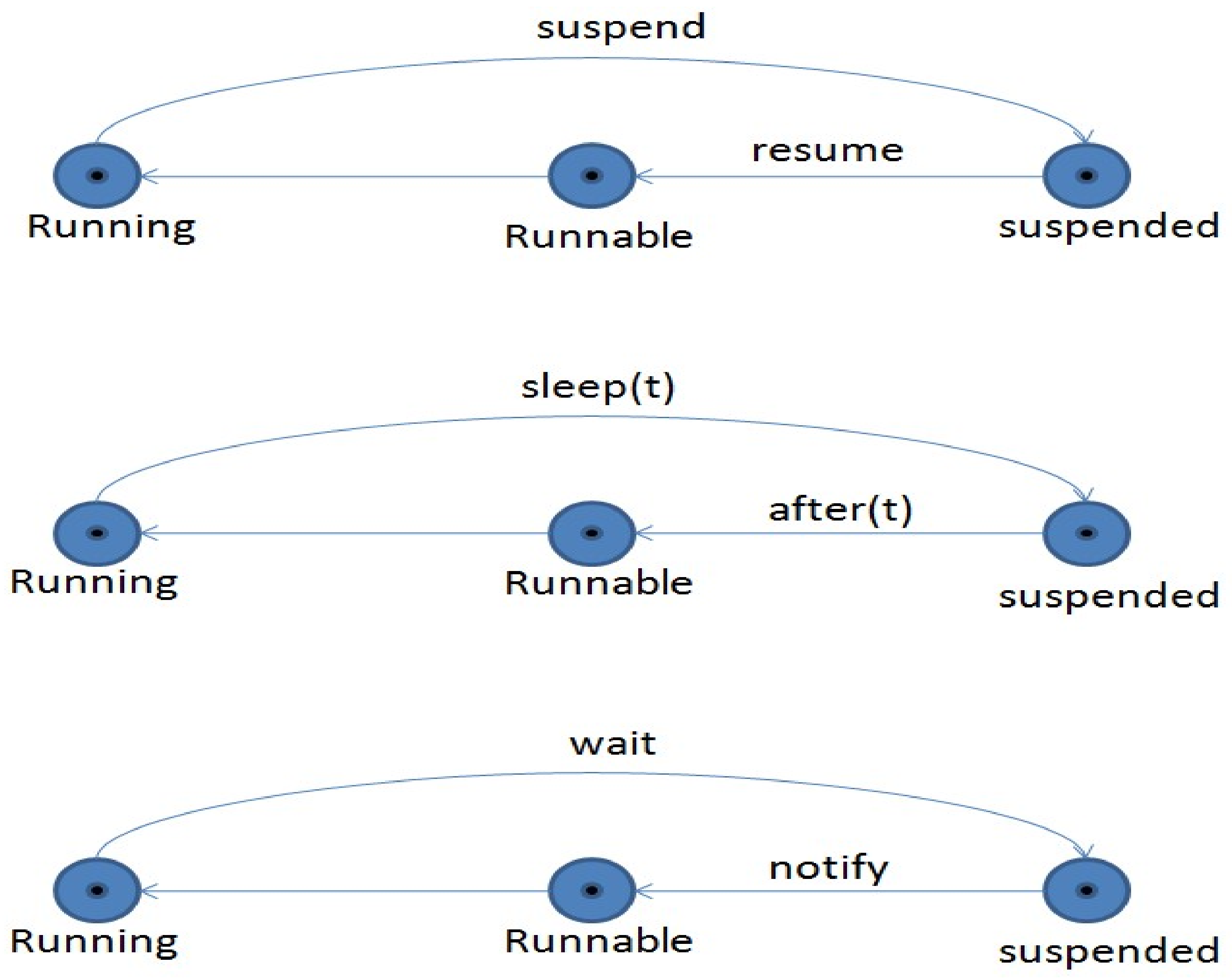
- It means that thread is now ready for running and its waiting to give control.
- We can move control to another thread by yield() method.



Relinquishing control using yield() method

Running State

- It means thread is in its execution mode becaause the control of cpu is given to that particular thread.
- It can be move in three different situation from running mode.



Blocked State

- A thread is called in Blocked State when it is not allowed to entering in Runnable State or Running State.
- It happens when thread is in waiting mode, suspended or in sleeping mode.

Dead State

- When a thread is completed executing its run() method the life cycle of that particular thread is end.
- We can kill thread by invoking stop() method for that particular thread and send it to be in Dead State.

Java Thread Priority

- Each java thread has its own priority which decides the order of thread to be schedule.
- The threads of equal priority will be given same treatment by java scheduler. And they will follow the FCFS (First Come First Serve) algorithm.
- User can also set the priority of thread by using the setPriority() method as follow:
- **ThreadName.setPriority(int Number);**

- Here the number is integer value between 1 to 10, Here 1 is minimum priority 10 is maximum priority.
- The Thread class defines few priority constants:

➢ MIN_PRIORITY = 1

➢ NORM_PRIORITY = 5

➢ MAX_PRIORITY = 10

- In any Thread the default priority is NORM_PRIORITY
- In multithreading by assigning priority we can answer an input as quickly as we want.
- Whenever more than one threads are ready to run java system select the highest priority thread and execute it
- If another thread of higher priority comes the running thread will be preempted by the incoming thread and current thread will move to runnable state.

```
class A extends Thread
{

    public void run()
    {
        System.out.println("ThreadA strated");
        for(int i = 1; i<=5; i++)
        {
            System.out.println("\t From ThreadA i = " +i);
        }
        System.out.println("Exit from A");
    }
}
```

```
class B extends Thread
{
    public void run()
    {
        System.out.println("ThreadB strated");
        for(int j = 1; j<=5; j++)
        {
            System.out.println("\t From ThreadB j = " +j);
        }
        System.out.println("Exit from B");
    }
}
```

```
public class Thread_Priority
{
    public static void main(String[] args)
    {
        A threadA = new A();
        B threadB = new B();
        threadA.setPriority(Thread.MIN_PRIORITY);
        threadB.setPriority(threadA.getPriority()+3);

        System.out.println("Start Thread A");
        threadA.start();

        System.out.println("Start Thread B");
        threadB.start();

        System.out.println("End of main Thread");
    }
}
```

```
Start Thread A
Start Thread B
End of main Thread
ThreadB strated
        From ThreadB j = 1
        From ThreadB j = 2
        From ThreadB j = 3
        From ThreadB j = 4
        From ThreadB j = 5
Exit from B
```

```
ThreadA strated
        From ThreadA i = 1
        From ThreadA i = 2
        From ThreadA i = 3
        From ThreadA i = 4
        From ThreadA i = 5
Exit from A
```

- Here You can see that we start ThreadA first but than also ThreadB completed its task first because of higher priority.
- But again its multithreading so we all know that output may be vary each time you run the program.

Thank you