Multithreading

Presented By :-

Shraddha Sheth

What is Thread?

- A thread is a single sequential flow of control within a program.
- Thread does not have its own address space but uses the memory and other resources of the process in which it executes.
- There may be several threads in one process
- The Java Virtual Machine (JVM) manages these and schedules them for execution.

What is Multithreading ?

- Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel.
- For example, one subprogram can display an animation on the screen while another may build the next animation to be displayed.
- A program that contains multiple flow of control is known as multithreaded program.

Cont..

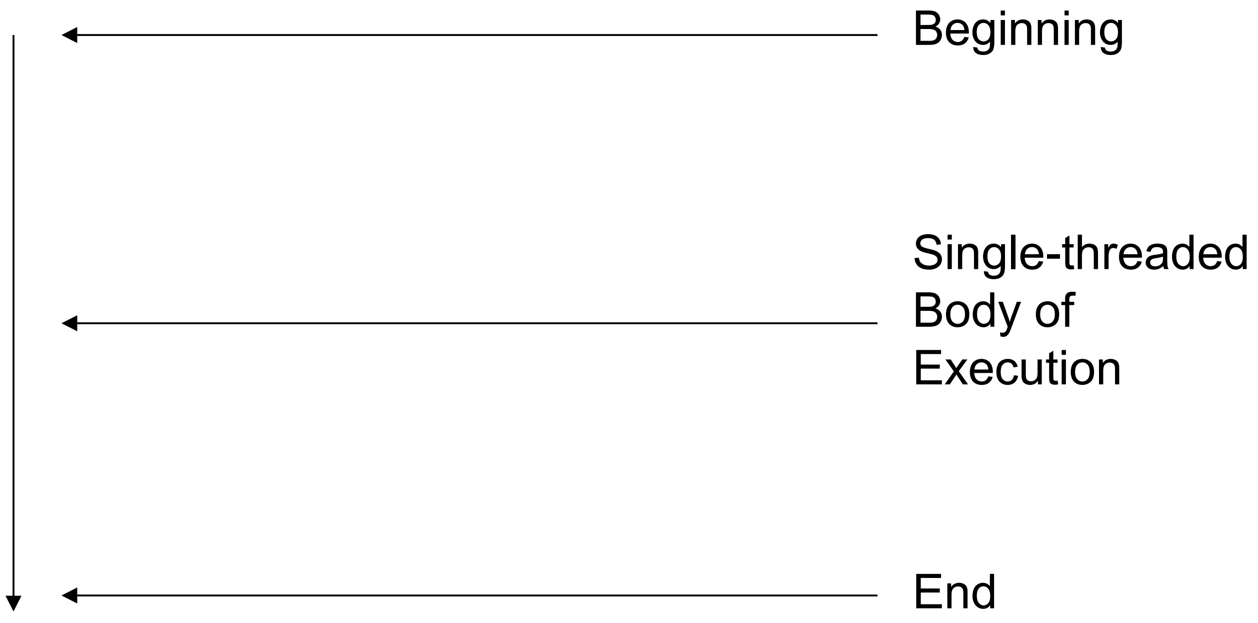
- Since threads in java are subprograms of a main application program and share the same memory space, they are known as lightweight threads or lightweight processes.
- ‘threads running in parallel’ does not really mean that they actually run at the same time.
- Since all the threads are running on a single processor, the flow of execution is shared between the threads. The java interpreter handles the switching of control between the threads in such a way that it appears they are running concurrently.

Cont..

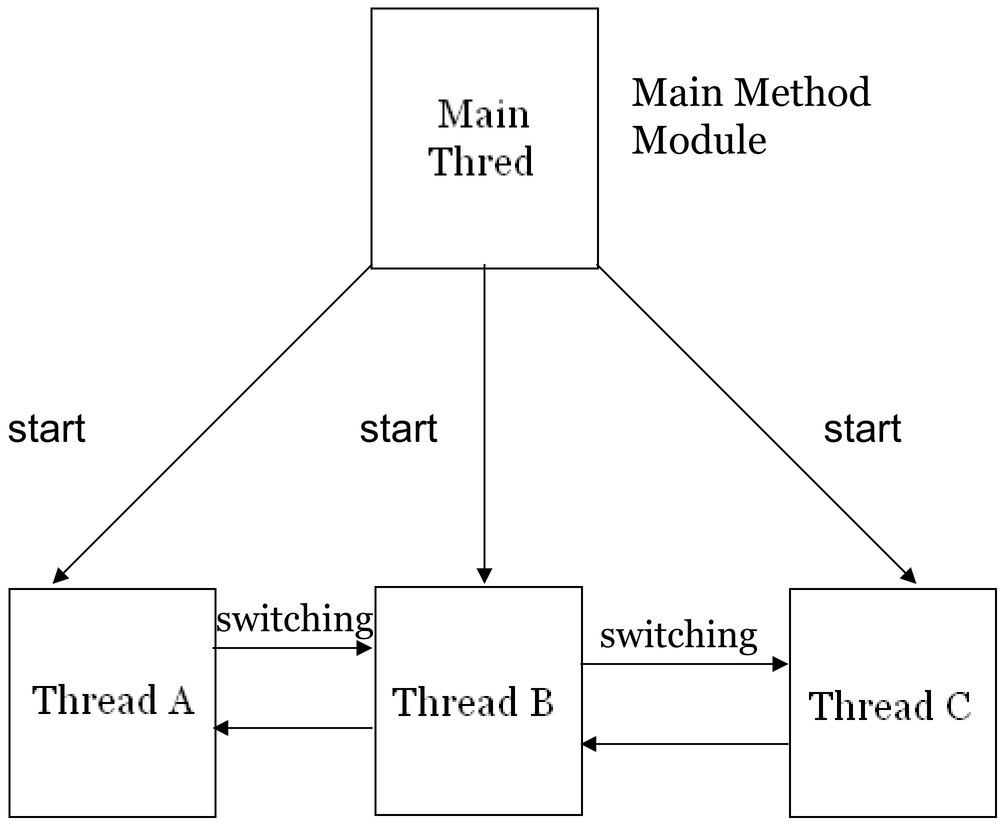
- Multithreading enables programmers to do multiple things at one time.
- For ex, we can send tasks such as printing into the background and continue to perform some other task in the foreground.
- Threads are extensively used in java-enabled browser such as HotJava. These browsers can download a file to the local computer, display a web page in the window, output another web page to a printer and so on.
- The ability of a language to support multithreads is referred to as concurrency.

A Single Threaded Program

```
class ABC
{
        ..........
        ...........
        ..........
        ...........
        ..........
        ...........
}
```



A Multithreaded Program



Thread Life Cycle

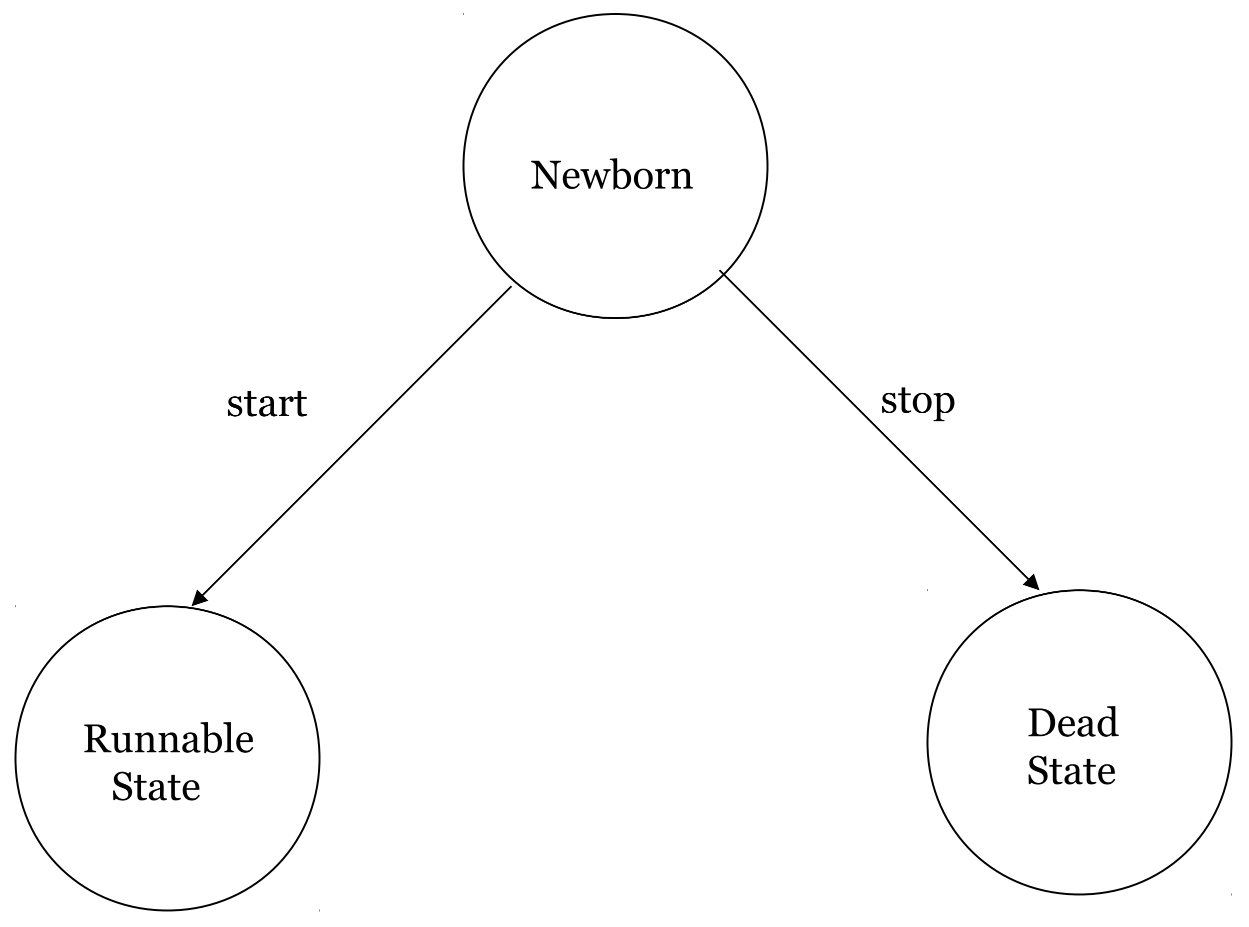
- During the life time of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable State
3. Running State
4. Blocked State
5. Dead State

1. Newborn State

- When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only one of the following with it:
  - Schedule it for running using start() method.
  - Kill it using stop() method
- If scheduled, it moves to the runnable state. If we attempt to use any other method at this stage, an exception will be thrown.

Scheduling a Newborn Thread



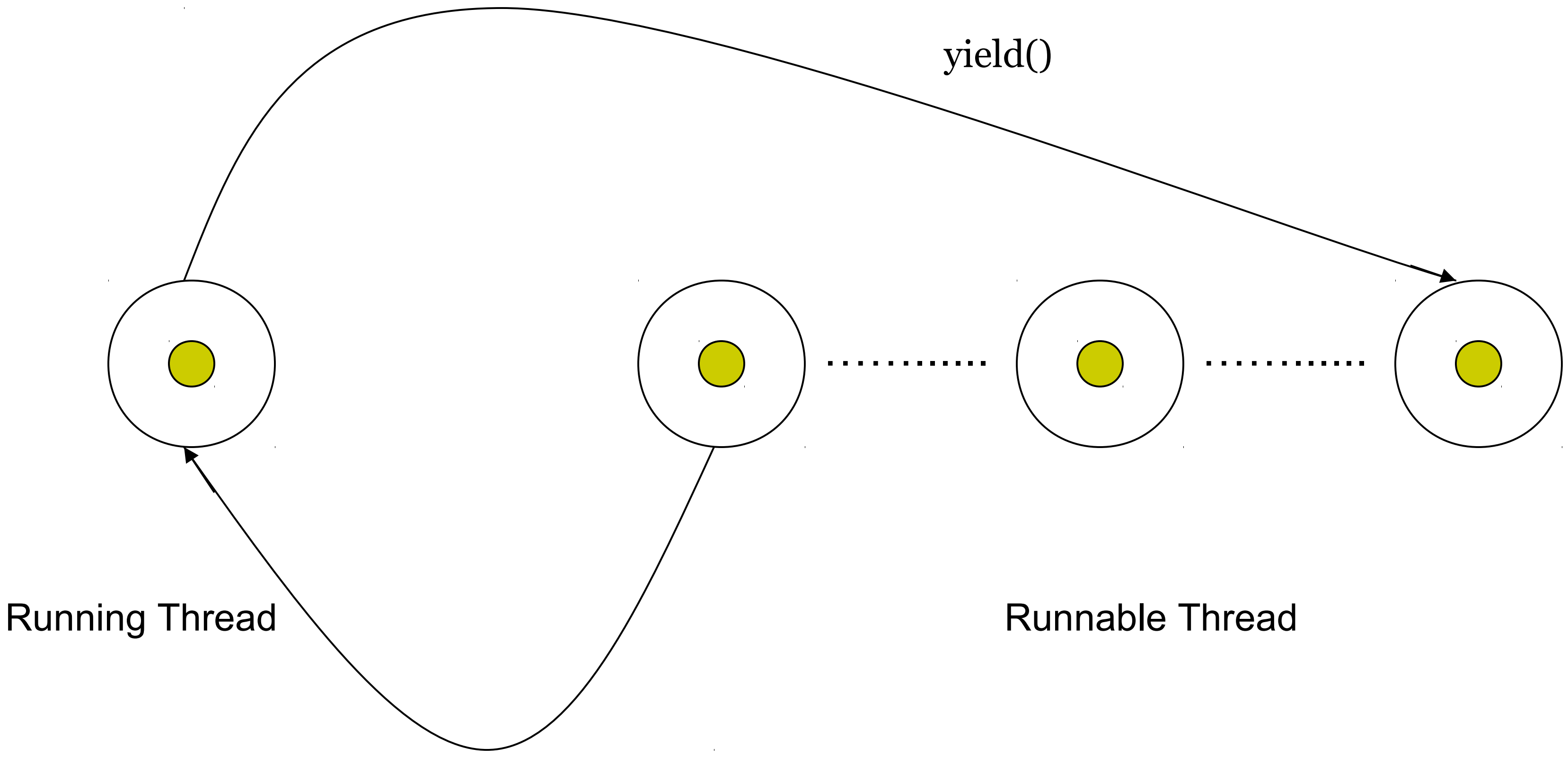
2. Runnable state (start())

- The runnable state means that the thread is ready for execution and is waiting for the availability of the processor.
- That is, the thread has joined the queue of threads that are waiting for execution.
- If all threads have equal priority, then they are given time slots for execution in round robin fashion. i.e. first-come, first-serve manner.

Cont..

- The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as *time-slicing*.
- If we want a thread to relinquish control to another thread of equal priority before its turn comes, we can do so by using the yield() method.

Releasing Control Using yield()

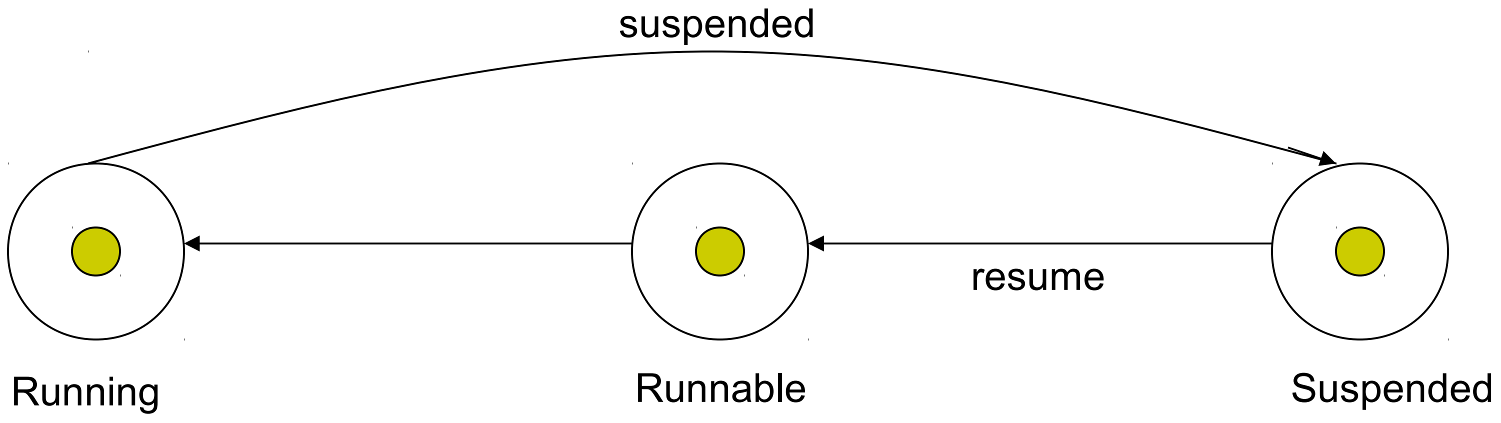


3. Running State

- Running means that the processor has given its time to the thread for its execution.
- The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.
- A running thread may relinquish its control in one of the following situations:

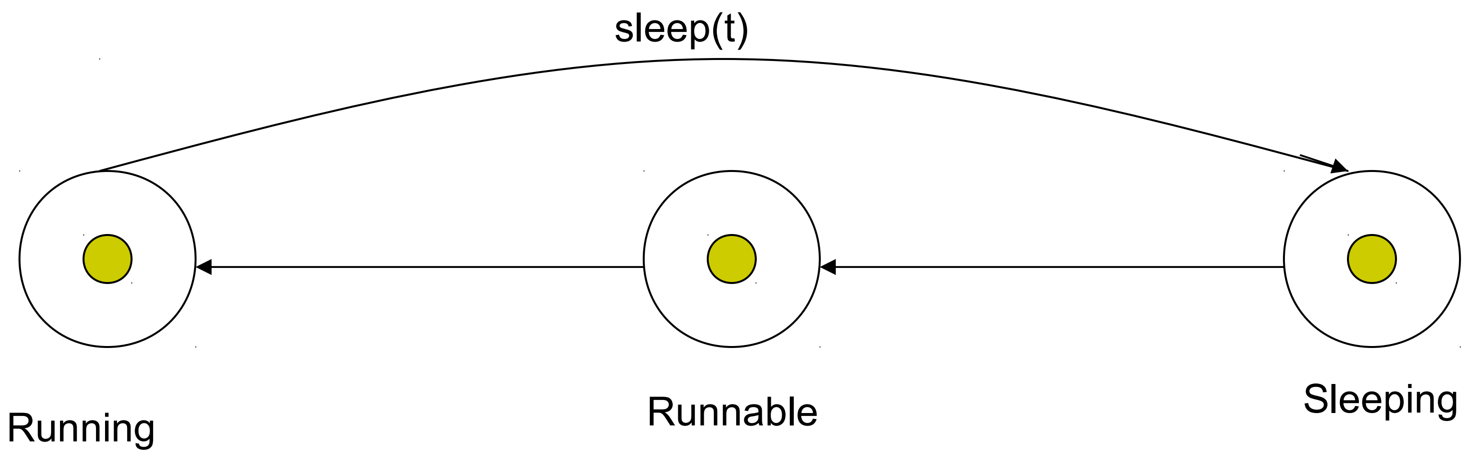
1. **Suspend() and resume() Methods**:-

This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it.

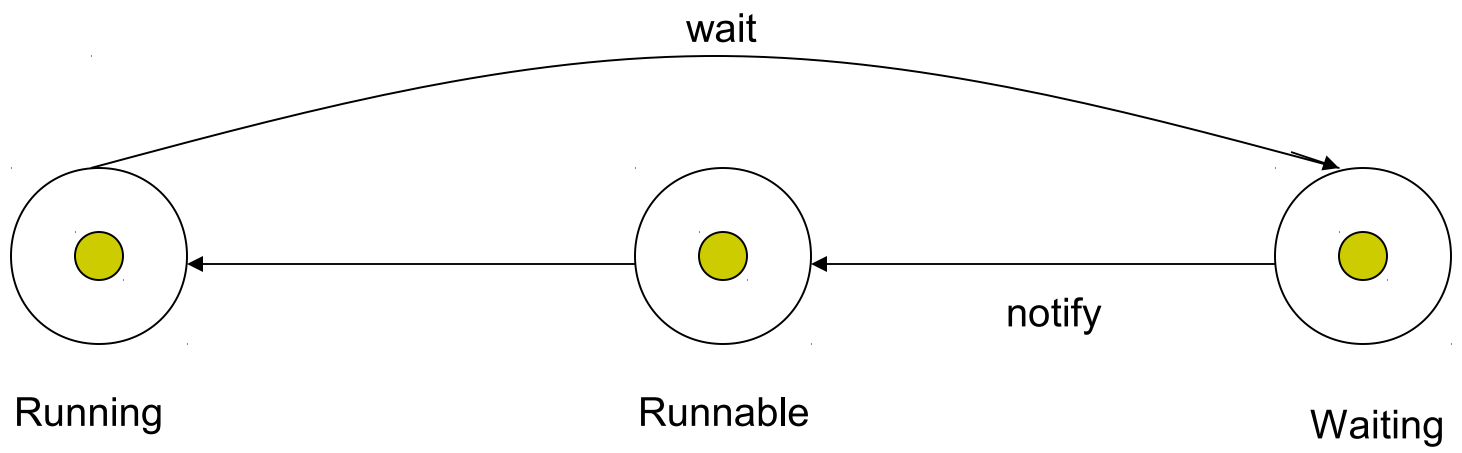


2. **Sleep() Method** :-

This means that the thread is out of the queue during this time period. The thread re-enter the runnable state as soon as this time period is elapsed.



3. **Wait() and notify() methods** :- blocked until certain condition occurs



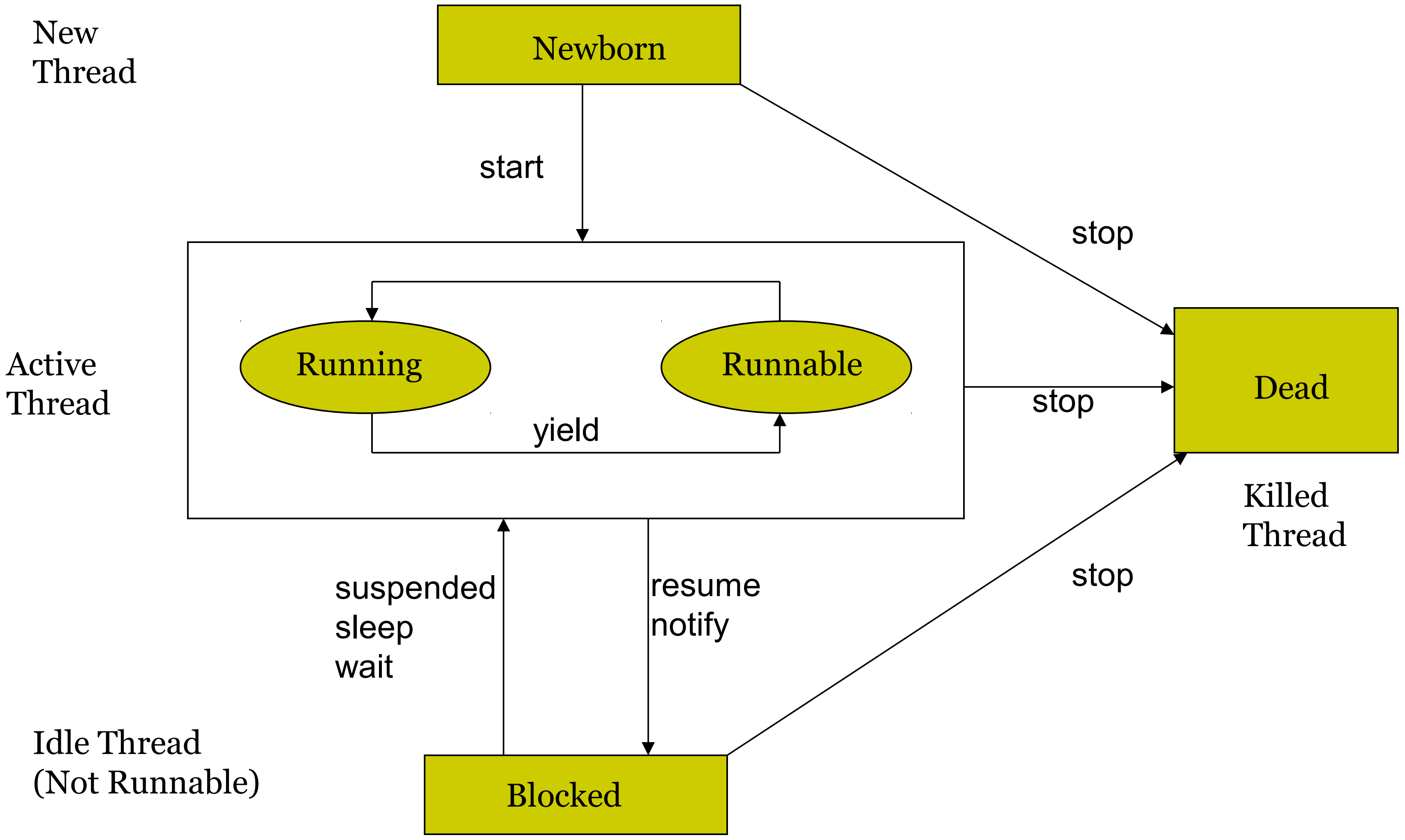
4. Blocked State

- A thread is said to be blocked when it is prevented form entering into the runnable state and subsequently the running state.
- This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements.
- A blocked thread is considered “ not runnable” but not dead and therefore fully qualified to run again.

5. Dead State

- A running thread ends its life when is has completed executing its run() method. It is a natural death.
- However, we can kill it by sending the stop message to it at any state thus causing a premature death to it.
- A thread can be killed as soon it is born, or while it is running, or even when it is in “not runnable” (blocked) condition.

State diagram of a thread



join() method :-

- The join method allows one thread to wait for the completion of another.
- If t is a Thread object whose thread is currently executing, t.join(); causes the current thread to pause execution until t's thread terminates.
- Overloads of join allow the programmer to specify a waiting period. However, as with sleep, join is dependent on the OS for timing, so you should not assume that join will wait exactly as long as you specify.
- Like sleep, join responds to an interrupt by exiting with an InterruptedException Example

Creating Threads

- Thread class in the java.lang package allows you to create and manage threads. Each thread is a separate instance of this class.
- A new thread can be created in two ways:
  1. By extending a thread class
  2. By implementing an interface

Extending the Thread class

- We can directly extend the Thread class

```
class Threadx extends Thread
{
    public void run()
    {
       //logic for the thread
    }
}
```

- The class ThreadX extends Thread. The logic for the thread is contained in the run() method.

Cont..

- It can create other objects or even initiate other threads.

Example

Implementing Runnable Interface

- Declare a class that implements the Runnable interface.
- This method declares only one method as shown here:

public void run();

```
class RunnableY implements Runnable
{
        Public vod run()
        {
                // logic for thread
        }
}
```

Cont..

- The application can start an instance of the thread by using the following code:

```
RunnableY ry = new RunnableY();
ThreadY ty= new Thread(ry);
Ty.start();
```

- 1st line instantiate the RunnableY class.
- 2nd line instantiate the Thread class. A reference to the RunnableY object is provided as the argument to the constructor.
- Last line starts the thread.

Example

Stopping a Thread

- ✓ Whenever we want to stop a thread form running further, we may do so by calling its stop() method like:

```
ty.stop();
```

- ✓ The stop() method may be used when the premature death of a thread is desired

Blocking a Thread

- ✓ A thread can also be temporarily suspended or blocked form entering into the runnable and subsequently running state by using either of the following thread methods:
  - Sleep() – blocked for a specified time.
  - Suspend() – blocked until further orders.
  - Wait() – blocked until certain condition occurs.

Cont..

- ✓ These methods cause the thread to go into the blocked state. The thread will return to the runnable state when the specified time is elapsed in the case of sleep(), the resume() method is invoked in the case of suspend(), and the notify() method is called in the case of wait().

Thread class

- ✓ Some of the constructors for Thread are as follows:
  - Thread()
  - Thread(Runnable r)
  - Thread(Runnable r, String s)
  - Thread(String s)
- ✓ Here, r is a reference to an object that implements the Runnable interface and s is a String used to identify the thread.

Methods of Thread class

• **Method**	**Description**
1. Thread currentThread()	returns a reference to the current thread
2. Void sleep(long msec) InterruptedException	causes the current Throws thread to wait for msec milliseconds
3. Void sleep(long msec, int nsec) InterruptedException	causes the current Throws to wait for msec milliseconds plus nsec nanoseconds
4. Void yield()	causes the current thread to yield control of the processo to other

Cont..

5. String getName()	returns the name of the thread.
6. Int getPriority()	returns the priority of the thread
7. Boolean isAlive()	returns true if this thread has been started and has not Yet died. Otherwise, returns false.
8. Void join() Throws InterruptedException	causes the caller to wait until this thread dies.
9. Void join(long msec) Throws InterruptedException	causes the caller to wait a max of msec until this thread dies. if msec is zero, there is no limit for the wait time.

Cont..

10. Void join(long msec, int nsec) Throws Interruptedexception	causes the caller to wait a max of msec plus nsec until this thread dies. If msec plus nsec is zero, there is no limit for the wait time.
11. Void run()	comprises the body of the thread. This method is overridden by subclasses.
12. Void setName(String s)	sets the name of this thread to s.
13. Void setPriority(int p)	sets the priority of this thread to p.
14. Void start()	starts the thread
15. String toString()	Returns the string equivalent of this thread.

Example

Thread Priority

- In java, each thread is assigned a priority, which affects the order in which it is scheduled for running.
- Java permits us to set the priority of a thread using the setPriority() method as follows:

  ThreadName.setPriority(intNumber);
- The intNumber may assume one of these constants or any value between 1 and 10.
- The intNumber is an integer value to which the thread's priority is set. The Thread class defines several priority constants:
  - MIN_PRIORITY=1
  - NORM_PRIORITY=5
  - MAX_PRIORITY=10
- The default setting is NORM_PRIORITY. Example

Synchronization

- ✓ When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.
- ✓ This can be done in two ways. First, a method can be synchronized by using the synchronized keyword as a modifier in the method declaration.

Cont..

- ✓ When a thread begins executing a synchronized instance method, it automatically acquires a lock on that object.
- ✓ The lock is automatically relinquished when the method completes.
- ✓ Only one thread has this lock at any time.
- ✓ Therefore, only one thread may execute any of the synchronized instance methods for that same object at a particular time.

Cont..

- ✓ Another way to synchronize access to common data is via a synchronized statement block. This has the following syntax:

```
synchronized(obj)
{
      //statement block
}
```

Example

Dead Lock

- ✓ Deadlock is an error that can be encountered in multithreaded programs.
- ✓ It occurs when two or more threads wait for ever for each other to relinquish locks.
- ✓ Assume that thread1 holds lock on object1 and waits for a lock on object2. thread2 holds a lock on object2 and waits for a lock on object1. neither of these threads may proceed. Each waits forever for the other to relinquish the lock it needs.

Cont..

- ✓ Deadlock situations can also arise that involve more than two threads. Assume that thread1 waits for a lock held by thread2. thread2 waits for a lock held by thread3. thread3 waits for a lock held by thread1.

Example

Thread Communication

- Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time.
- For example, consider the classic queuing problem, where one thread is producing some data and another is consuming it.
- In a polling system, the consumer would waste many CPU cycles while it waited for the producer to produce. Once the producer was finished, it would start polling, wasting more CPU cycles waiting for the consumer to finish, and so on. Clearly, this situation is undesirable.

- To avoid polling, Java includes an elegant inter-process communication mechanism via the **wait( )**, **notify( )**, and **notifyAll( )** methods. These methods are implemented as **final** methods in **Object**, so all classes have them.
- All three methods can be called only from within a **synchronized** method.

Example

Thank You..