SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23CSB101- OBJECT ORIENTED PROGRAMMING

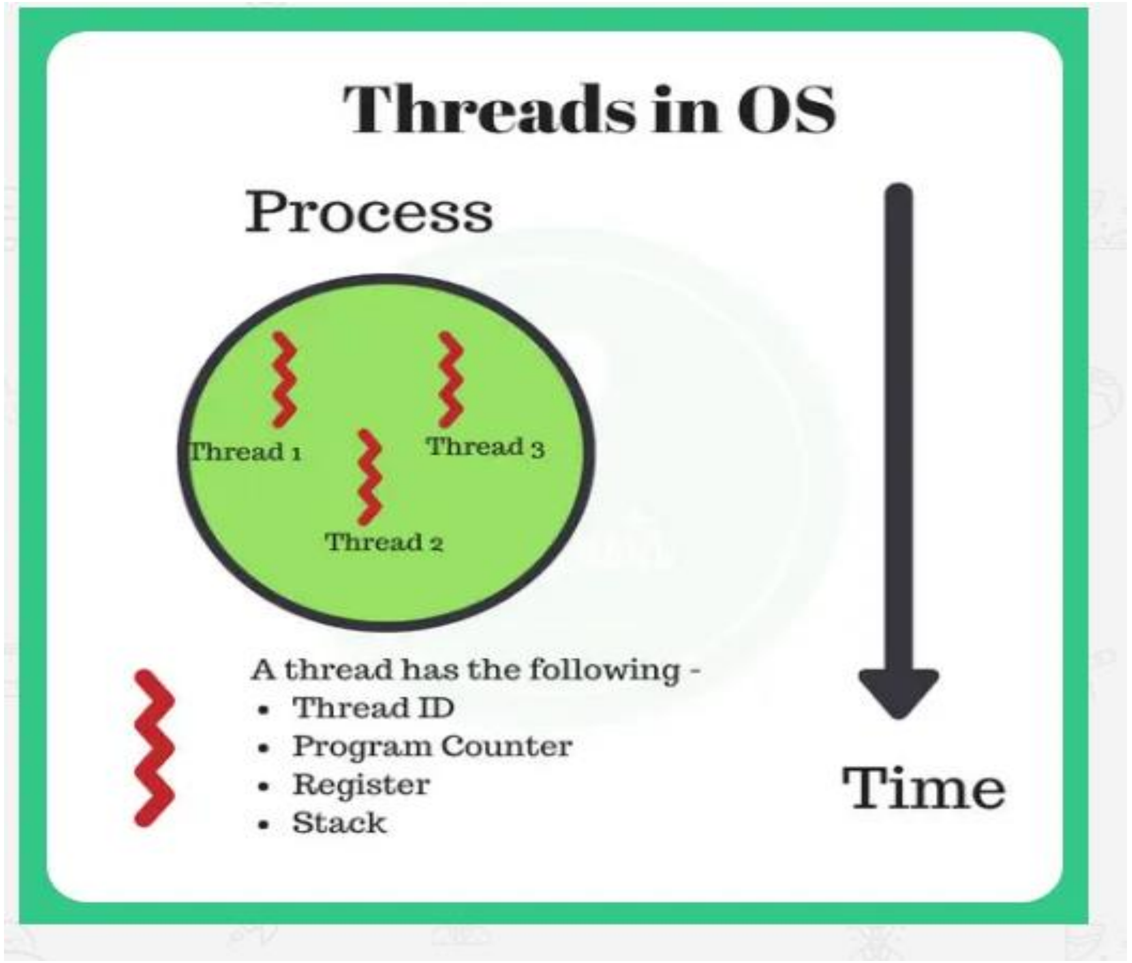
I YEAR /II SEMESTER

Unit III – EXCEPTION HANDLING AND MULTITHREADING

Topic : MULTITHREADED PROGRAMMING

Thread

- A thread is a **lightweight sub-process** that defines a separate path of execution. It is the smallest unit of processing that can run concurrently with other threads of the same process.
- **Multithreading** is a **technique** of executing more than one thread, performing different tasks, simultaneously.
- **Process**: Process is a **heavy weight program**. Each process has a complete set of its own variables. Use **IPC** to communicate between processes.



A thread has the following -

- Thread ID
- Program Counter
- Register
- Stack

THREAD

Advantages of Threads / Multithreading:

1. Threads are light weighted.
2. Threads share the same address space for both data and code.
3. Context switching between threads is less expensive.
4. Low computation and Communication cost.
5. Threads allow different tasks to be performed concurrently.
6. Multithreading allows efficient utilization of system resources.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. It is used to maximize CPU utilization.

Two types:

Process-based Multitasking (Multiprocessing):

Executing two or more programs concurrently.

Thread-based Multitasking (Multithreading):-

Single program can perform two or more tasks simultaneously.

Thread Model / Thread Life Cycle

Different states of a Thread

1. New State

2. Runnable State

3. Running State

4. Waiting/Timed Waiting/Blocked state

5. Terminated State/ dead state

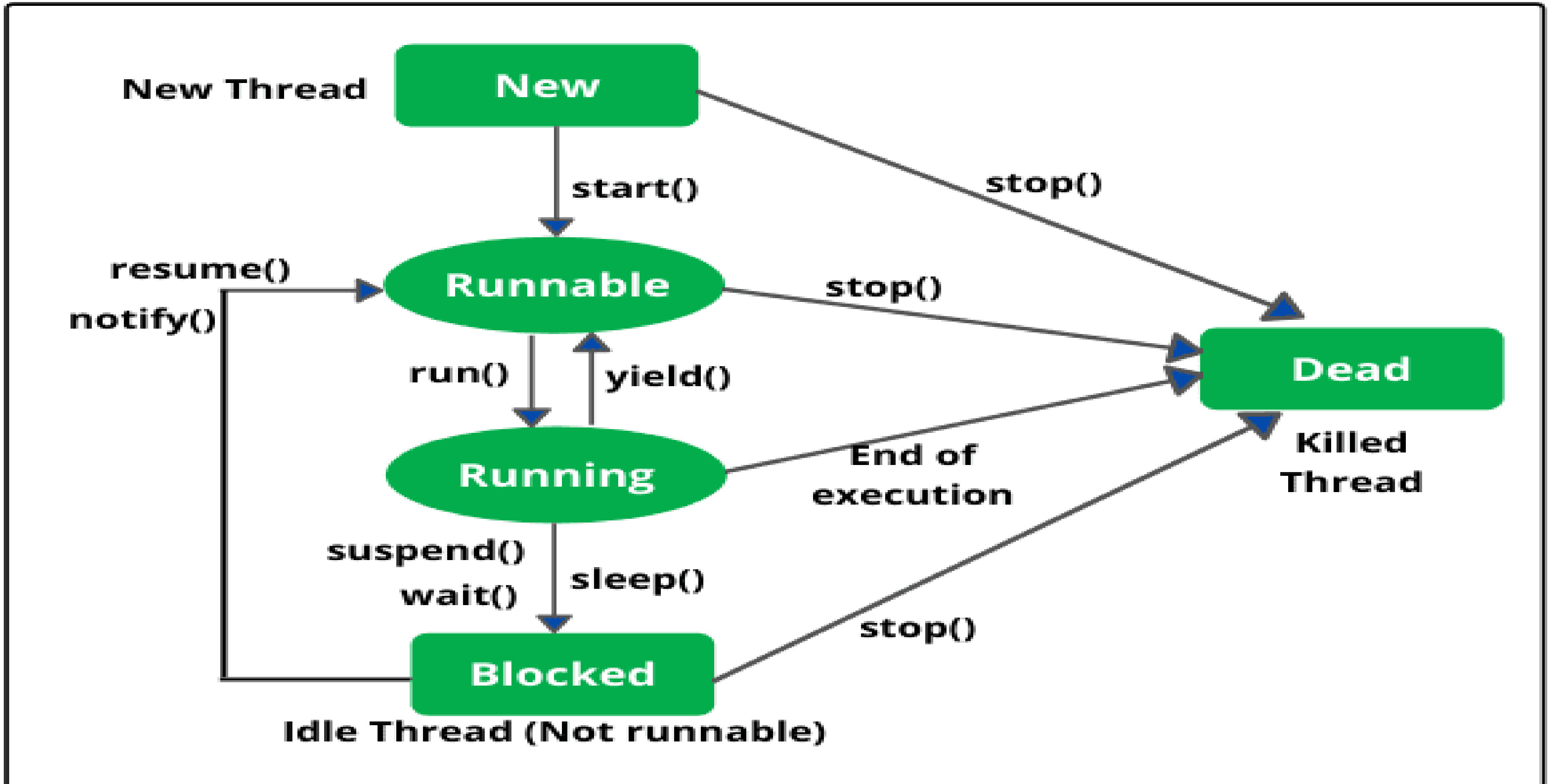


Fig: State Transition Diagram of a Thread

Thread Model / Thread Life Cycle

1. <u>New State</u>: start()

A **new thread (**born thread) **begins** here. It **remains in this state** until the program starts the thread by calling **start()** method, which **places** the thread in the **runnable** state.

Example:

```
Thread myThread=new Thread();
```

Only start() and stop() methods can be called. **Calling** any **other methods causes an IllegalThreadStateException**.

Thread Model / Thread Life Cycle

2. Runnable State: run()

After creation, the thread becomes runnable or running by calling the **run()** method. A thread starts executing its task.

Example:

```
myThread.start();
```

This creates the system resources necessary to run the thread, schedules the thread to run and **calls** the thread’s **run()** method.

Thread Model / Thread Life Cycle

3. Running state:

- Thread **scheduler** schedules thread to from **runnable to running state**. In running state **thread starts executing** by entering run() method.
- Thread scheduler selects thread from the runnable pool **on basis of priority**, if priority of two threads is same, threads are scheduled in unpredictable manner.
- When threads are in running state, **yield()** method can **make thread to go in Runnable state**.

Thread Model / Thread Life Cycle

4. Waiting/Timed Waiting/Blocked State : wait()

- **Waiting State:** Sometimes one thread has to undergo in waiting state because another thread starts executing. A runnable thread can be moved to a waiting state by calling the **wait()** method.
- A call to **notify() and notifyAll()** may bring the thread from waiting state to runnable state.

Thread Model / Thread Life Cycle

<u>4. Waiting/Timed Waiting/Blocked State :</u> sleep()

- **Timed Waiting:** A runnable thread can enter the timed waiting state for a specified interval of time by calling the **sleep()** method. After the **time elapses**, the thread in waiting state enters into the **runnable state**.

Code:

```
try {
        Thread.sleep(3*60*1000);// thread sleeps for 3 minutes
  }
catch(InterruptedException ex) { }
```

Thread Model / Thread Life Cycle

4. Waiting/Timed Waiting/Blocked State : suspend()

- **Blocked State:** When a particular thread issues an I/O request, then operating system moves the thread to blocked state until the I/O operations gets completed. This can be achieved by calling **suspend()** method.
- After the **I/O completion**, the thread is **sent back to the runnable** state.

Thread Model / Thread Life Cycle

5. Terminated State:

A **runnable thread** enters the terminated state when,

(i) It completes its task (when the run() method has finished)

```
public void run() { }
```

(ii) Terminates (when the stop() is invoked)

```
myThread.stop();
```

Thread Model / Thread Life Cycle

New : A thread begins its life cycle in the new state. It remains in this state until the **start()** method is called on it.

Runnable : After invocation of start() method on new thread, the thread becomes runnable. **Runnable interface**

Running : A thread is in running state if the thread scheduler has selected it. **run()**

Waiting : A thread is in waiting state if it waits for another thread to perform a task. In this stage the thread is still alive. **wait() / suspend()**

Terminated : A thread enter the terminated state when it complete its task. **stop()/kill()**

Program

1. Define the Even class (implements Runnable interface):

2.Define the Odd class (implements Runnable interface):

3.Define the Generate class (extends Thread):

4.In the main() method of Multithread class:

4a..Create an instance of the Generate class.

4b.Start the Generate thread, which in turn generates random numbers and creates even/odd threads.

Program

```
  import java.util.*;
class Even implements  Runnable
  {
      public int x; public Even(int x)
  {
      this.x = x;
  }
   public void run()
  {
  System.out.println("New  Thread "+ x +" is EVEN  and Square of " + x + " is: " + x * x);
  }
  }
```

Cont…

Program

```
class Odd implements Runnable
{
public int x; public Odd(int x)
{
        this.x = x;
}
public void run()
{
System.out.println("New Thread "+ x +" is ODD and Cube of " + x + " is: " + x * x * x);
}
}
```

Cont…

Program

```
class Generate extends Thread
{
public void run()
{
    int num = 0;
    Random r = new Random();
    try
     {
         for (int i = 0; i < 5; i++)
     {
         num = r.nextInt(100);
          System.out.println("Main Thread Generates Random Integer: " + num);
```

Cont…

Program

```
 if (num % 2 == 0)
{
     Thread t1 = new Thread(new Even(num));
     t1.start();
}
else
{
      Thread t2 = new Thread(new Odd(num));
      t2.start();
}
Thread.sleep(1000);
System.out.println("..................................................");
}   // for loop
}  // try block
```

Cont…

Program

```
catch (Exception ex)
{
    System.out.println(ex.getMessage());
}  // catch
}  // run()
} // class Generate
public class Multithread
{
public static void main(String[]  args)
{
Generate g = new Generate();
g.start();
  } //main
} //Multithread
```

Program

OUTPUT:

```
Main Thread Generates Random Integer: 33
New Thread 33 is ODD and Cube of 33 is: 35937

Main Thread Generates Random Integer: 72
New Thread 72 is EVEN and Square of 72 is: 5184

Main Thread Generates Random Integer: 91
New Thread 91 is ODD and Cube of 91 is: 753571

Main Thread Generates Random Integer: 54
New Thread 54 is EVEN and Square of 54 is: 2916

Main Thread Generates Random Integer: 8
New Thread 8 is EVEN and Square of 8 is: 64
```

`

Thank you