SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
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 COMPUTER APPLICATIONS

23CAT606 – JAVA PROGRAMMING

I YEAR II SEM

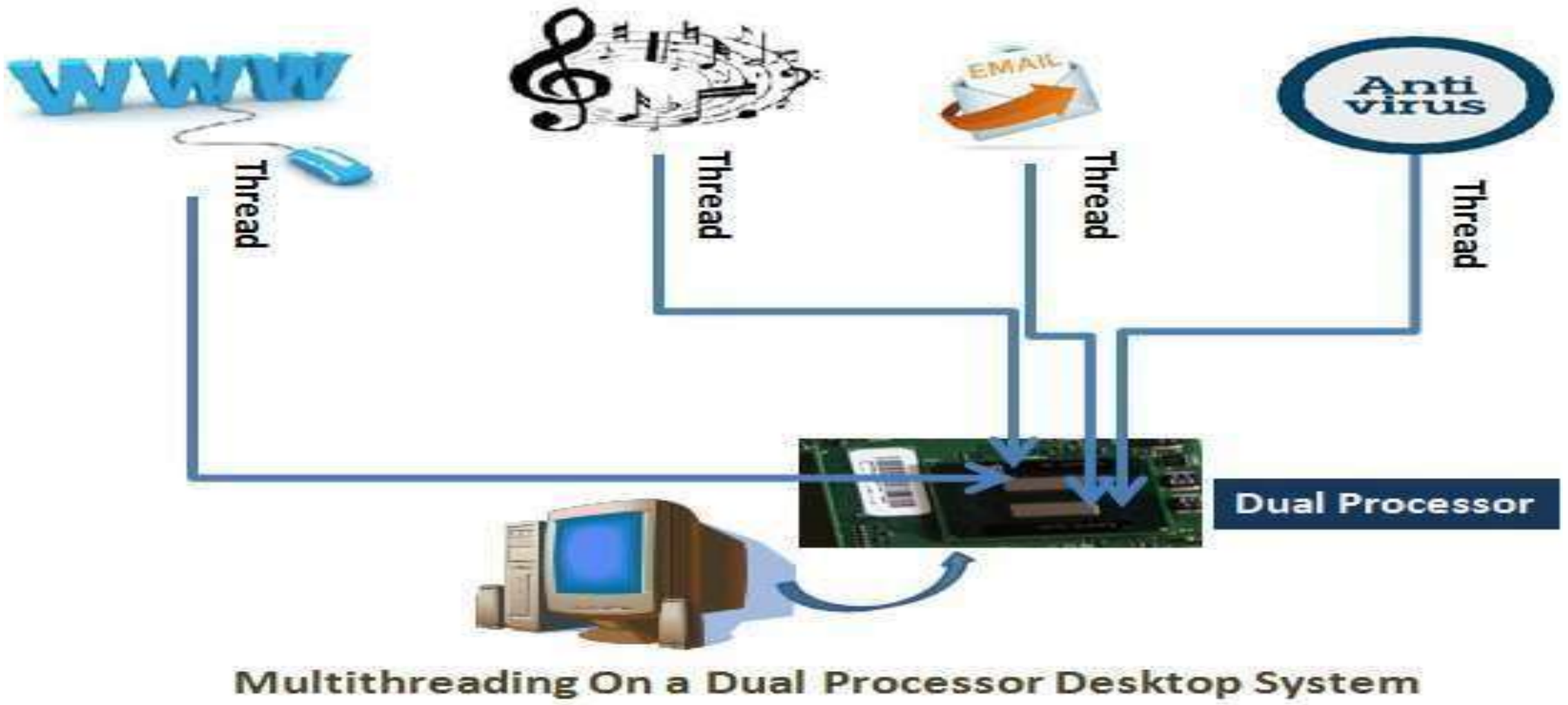
UNIT III – NETWORKING AND I/O PACKAGES

TOPIC 1 - MULTITHREADING

Agenda

- Introduction to Thread
- Creation of Thread
- Life cycle of Thread
- Stopping and Blocking a Thread
- Using Thread Methods
- Thread Priority
- Thread Synchronization
- DeadLock

MULTITHREADING



Multithreading On a Dual Processor Desktop System

INTRODUCTION TO THREAD

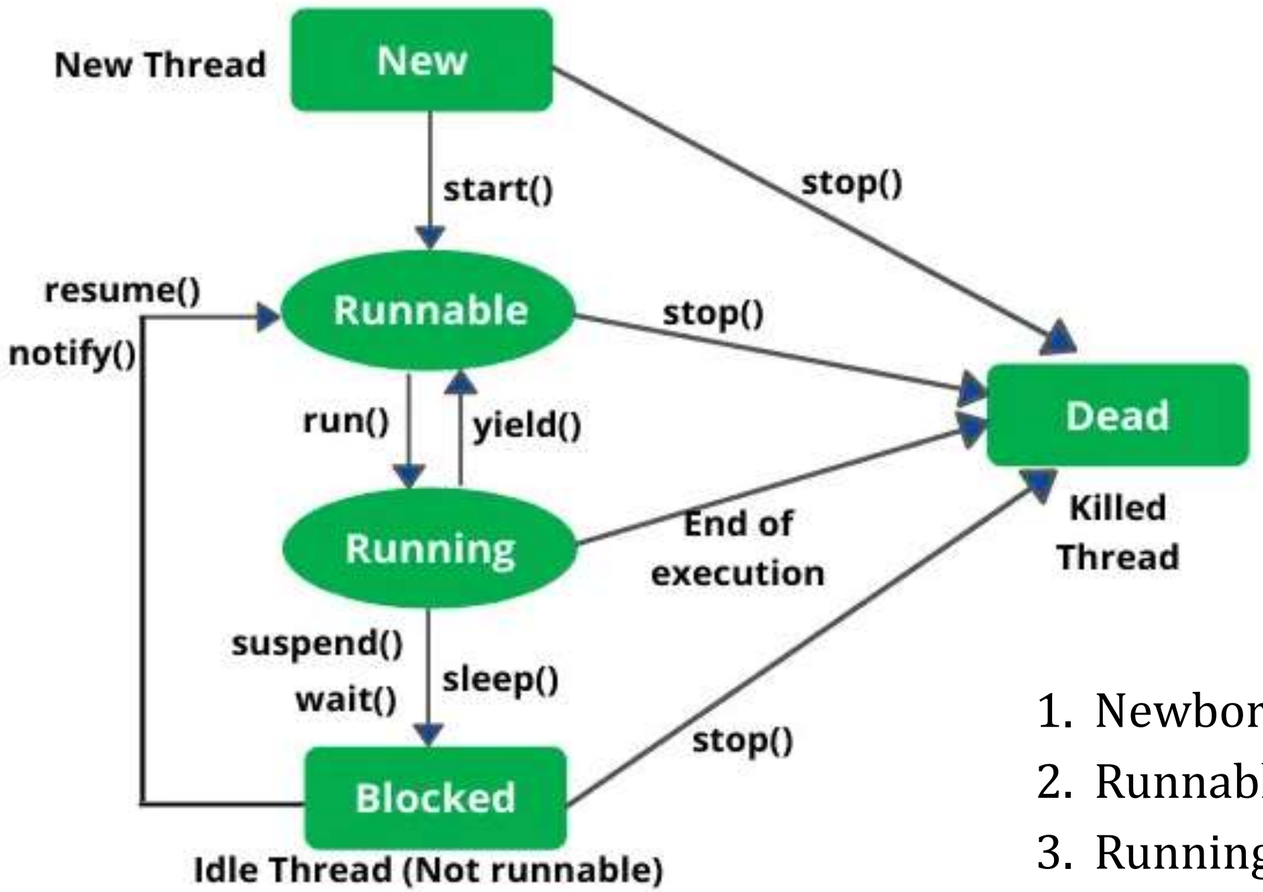
Process and **Thread** are two basic units of Java program execution.

1. **Process:** A process is a self contained execution environment and it can seen as a program or applications
2. **Thread:** It can be called lightweight process. Thread requires less resources to create and exists in the process. Thread shares the process resources.
3. **Multithreading** in java is a process of executing multiple processes simultaneously.
4. A program is divided into two or more subprograms, which can be implemented at the same time in parallel.
5. **Multiprocessing and multithreading**, both are used to achieve multitasking.
6. Java Multithreading is mostly used in **games, animation** etc.

Why Use Multithreading?

1. **Resource Sharing:** Threads share the same process's resources, such as memory, which makes inter-thread communication easier.
2. **Responsiveness:** Applications can remain responsive to user inputs, even if a part of it is waiting for a long operation to complete.
3. **Performance:** On multi-core processors, threads can run in parallel, leading to better resource utilization and faster application performance.

LIFE cycle of a thread



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

Java provides built-in support for multithreading via its **java.lang** package.

The primary classes and interfaces involved are:

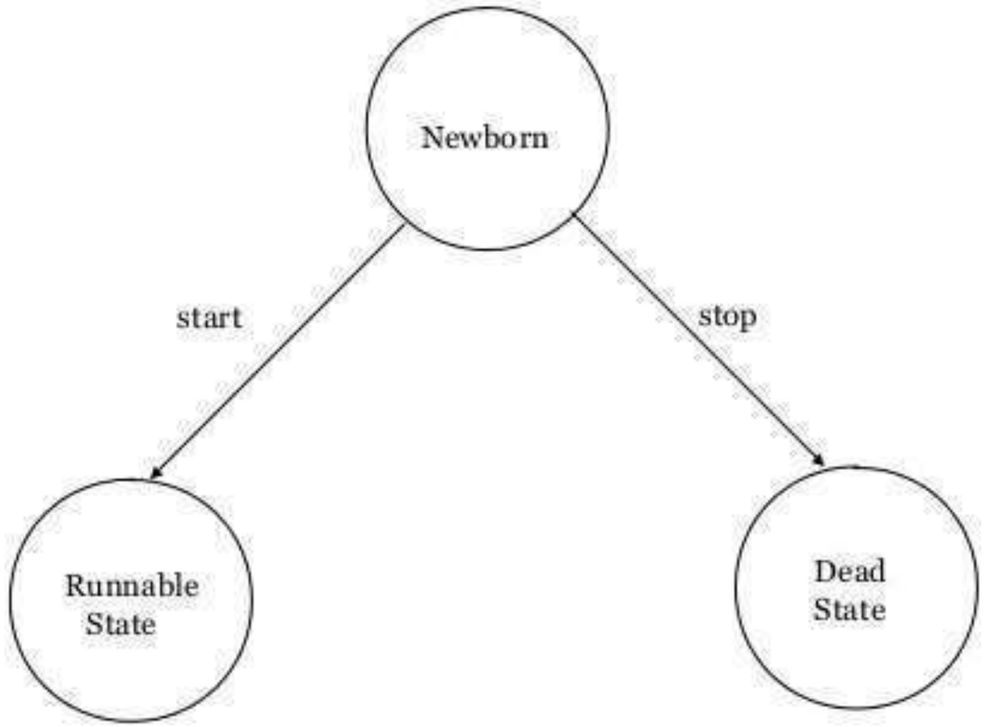
1.Thread class

2.Runnable interface

LIFE cycle of a thread

Newborn State:

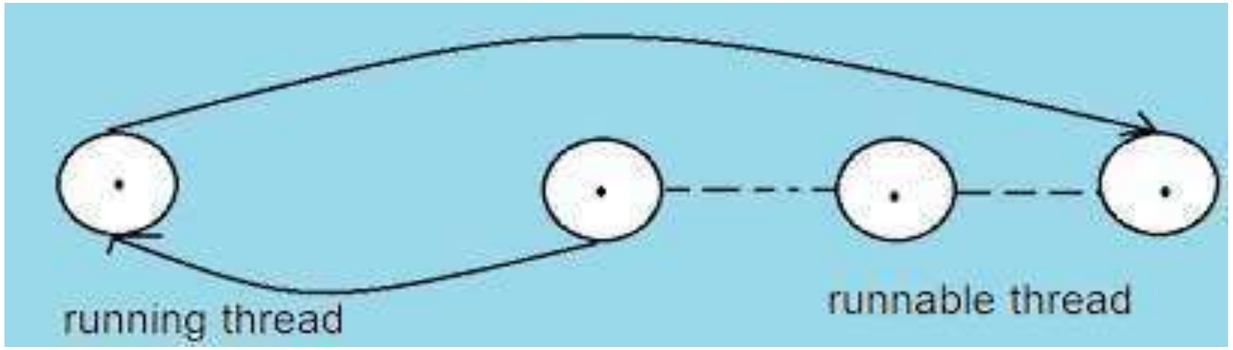
1. The thread is born and is said to be in newborn state.
2. The thread is not yet scheduled for running.
3. At this state, we can do only one of the following:
    1. Schedule it for running using **start() method.**
    2. Kill it using **stop() method.**



LIFE cycle of a thread

Runnable State:

1. The thread is ready for execution
2. Waiting for the availability of the processor.
3. The thread has joined the queue



Running State:

1. Thread is executing
2. The processor has given its time to the thread for its execution.
3. The thread runs until it gives up control on its own or taken over by other threads.

LIFE cycle of a thread

Blocked State:

1. A thread is said to be blocked
2. It is prevented to entering into the runnable and the running state.
3. This happens when the thread is
    1. Suspended: suspend()
    2. Sleeping: sleep()
    3. Waiting : wait() - in order to satisfy certain requirements.
4. A blocked thread is considered "**not runnable" but not dead** and therefore fully qualified to run again.

LIFE cycle of a thread

Dead State:

1. Every thread has a life cycle.
2. Natural Death: A running thread ends its life when it has completed executing its run( ) method. It is a natural death.
3. Premature Death: A thread can be killed in born, or in running, or even in "not runnable" (blocked) condition. This state is achieved when we invoke stop() method or the thread completes it execution.

Thread Methods

Method Signature	Description
String getName()	Retrieves the name of running thread in the current context in String format
void start()	This method will start a new thread of execution by calling run() method of Thread/runnable object.
void run()	This method is the entry point of the thread. Execution of thread starts from this method.
void sleep(int sleeptime)	This method suspend the thread for mentioned time duration in argument (sleeptime in ms)
void yield()	By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
void join()	This method used to queue up a thread in execution. Once called on thread, current thread will wait till calling thread completes its execution
boolean isAlive()	This method will check if thread is alive or dead

Program: Using Runnable interface

```
class Multi implements Runnable {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId() + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}
```

```
public class Multithread {

    public static void main(String[] args)
    {

        int n = 8; // Number of threads

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

            Multi obj = new Multi();

            obj.start();

        }

    }

}
```

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Program: Using Thread Class

```
class Multi extends Thread {
   public void run()
   {
      try {
         // Displaying the thread that is running
         System.out.println(
            "Thread " + Thread.currentThread().getId() + " is running");
      }
      catch (Exception e) {
         // Throwing an exception
         System.out.println("Exception is caught");
      }
   }
}
```

```
public class Multithread {

   public static void main(String[] args)
   {
      int n = 8; // Number of threads
      for (int i = 0; i < n; i++) {
         Multi obj = new Multi();
         obj.start();
      }
   }
}
```

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Thread Class vs Runnable Interface

1. If we extend the Thread class, our **class cannot extend** any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
2. We can **achieve basic functionality** of a thread by extending Thread class because it provides some inbuilt methods like **yield(), interrupt()** etc. that are not available in Runnable interface.
3. Using runnable will give you an **object that can be shared amongst multiple threads.**


Thank
you