Website

https://www.curiousinfotechsolutions.com

Multithreaded Programming

Java Programming

Day-22

Email id:
info@curiousinfotechsolutions.com

Syllabus Day-24

- **Brief**
  - ✓ **Java Swing**
    - ✓ Introduction
    - ✓ Difference between AWT and Swing
    - ✓ What is JFC
    - ✓ Hierarchy of Java Swing classes
    - ✓ Commonly used Methods of Component class
    - ✓ Java Swing Examples
    - ✓ Simple Java Swing Example

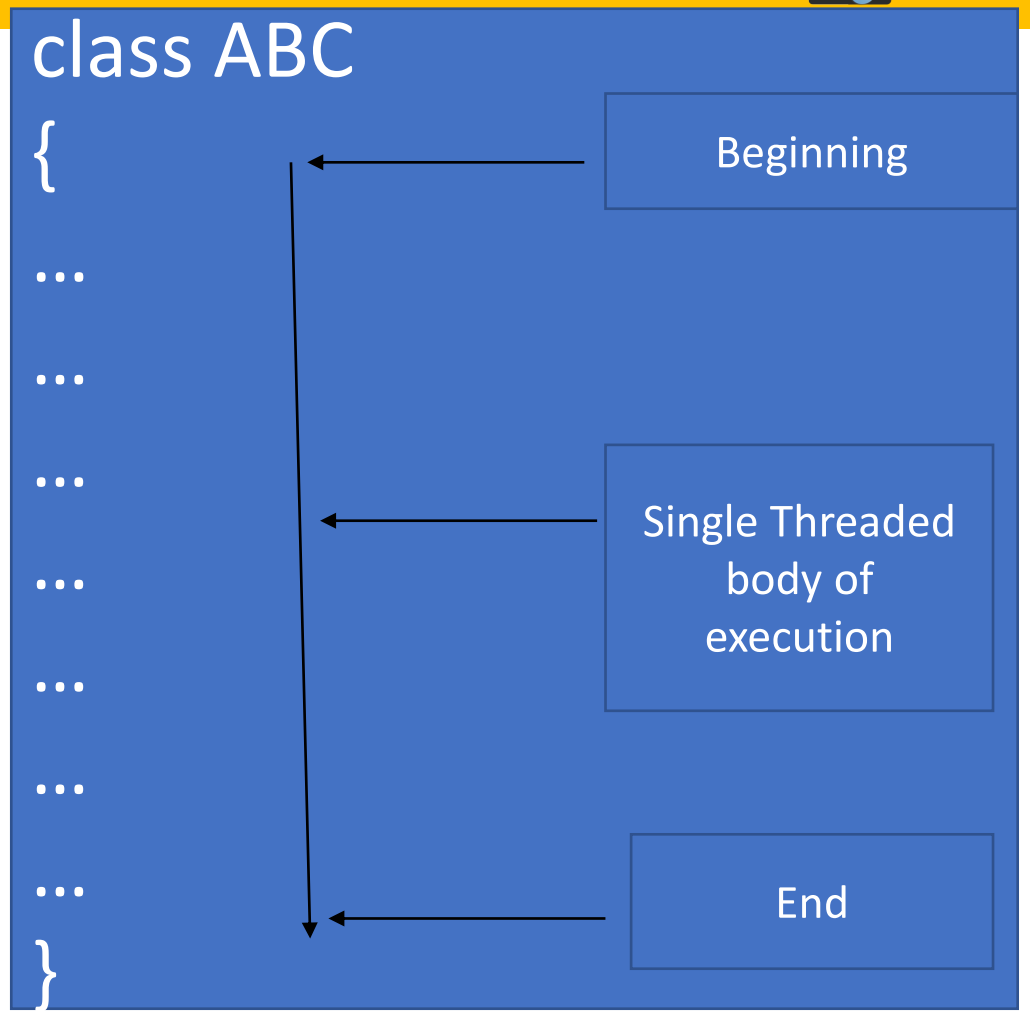
Introductions

- As Modern Operating Systems such as Windows 95 can *execute several program simultaneously*. This ability is known as **multitasking.** In system's terminology it is called **multithreading**.
- **Multithreading** is a *conceptual programming paradigm* where a **program (process)** is divided into two or more subprograms (processes) which can be implemented at the same time in **parallel.**
- **In most of our computers,** we have only a **single processor** and therefore, in reality, the processor is doing only one thing as a time. **However,** the **processor switches** between the processes so fast that it appears to human beings that all of them are being done simultaneously.

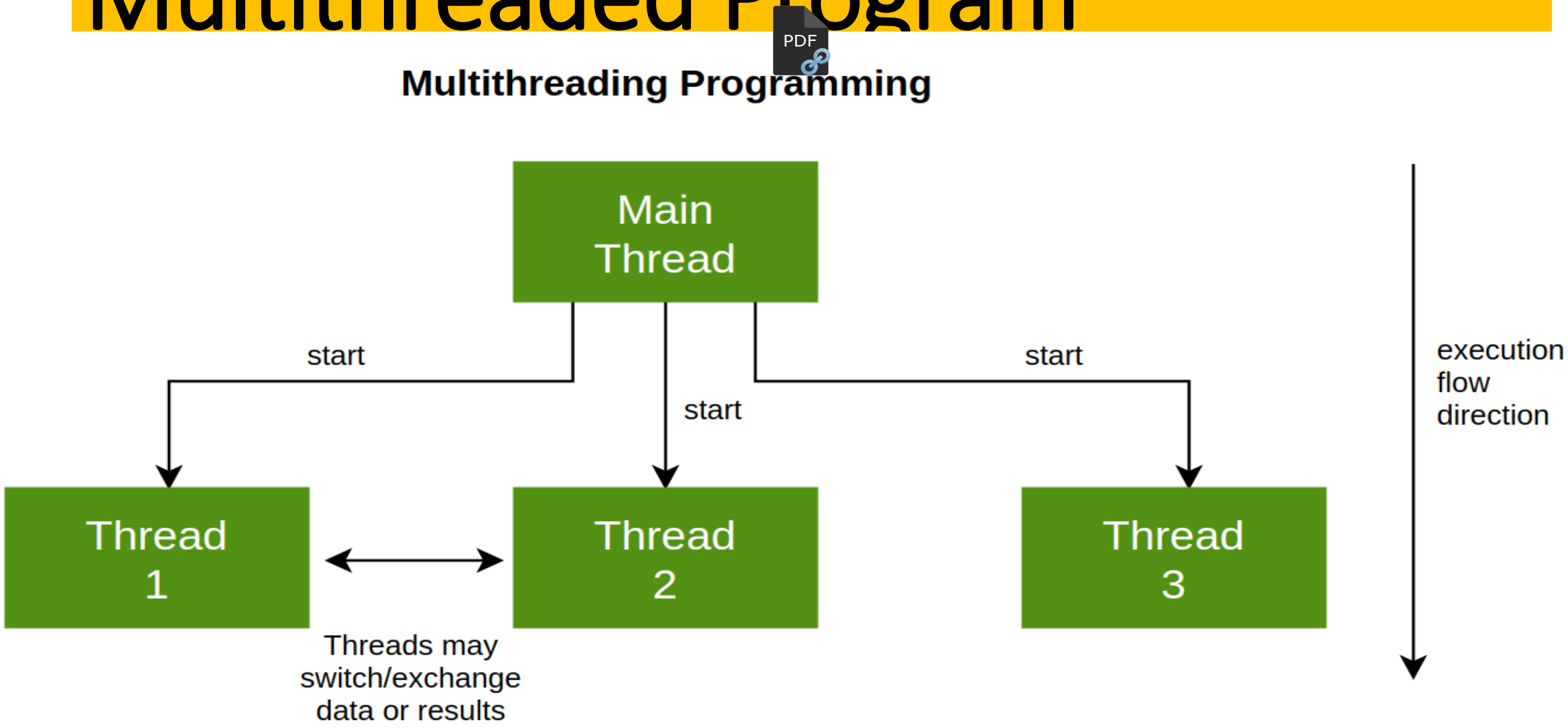
Contd..

- **Java programs** that we have seen and discuses so far contain only a **single sequential flow of control**. This is what happens when we execute a **normal program.** The program **begins**, **runs** through a sequence of executions, and finally **ends**. At any given point, there is only one statement under execution.
- **A thread** *is similar to a* **program** that *has a single flow of control*. It has a beginning, a body and an end, and execute commands sequentially. In fact, **all main programs** in our earlier examples can be called **single-threaded programs**. Every program will have at least one thread.

Single Threaded Program



Multithreaded Program



One Threaded Program


```
class A extends Thread
{
    public void run()
    {
        for(int i=1; i<=5; i++)
                System.out.println("\t From Thread A : i = "+ i);
        System.out.println("Exit From A");
    }
}
class Main
{
    public static void main (String[] args) {
        new A().start();
    }
}
```

```
From Thread A : i = 1
        From Thread A : i = 2
        From Thread A : i = 3
        From Thread A : i = 4
        From Thread A : i = 5
Exit From A
```

Multi-Threaded Program

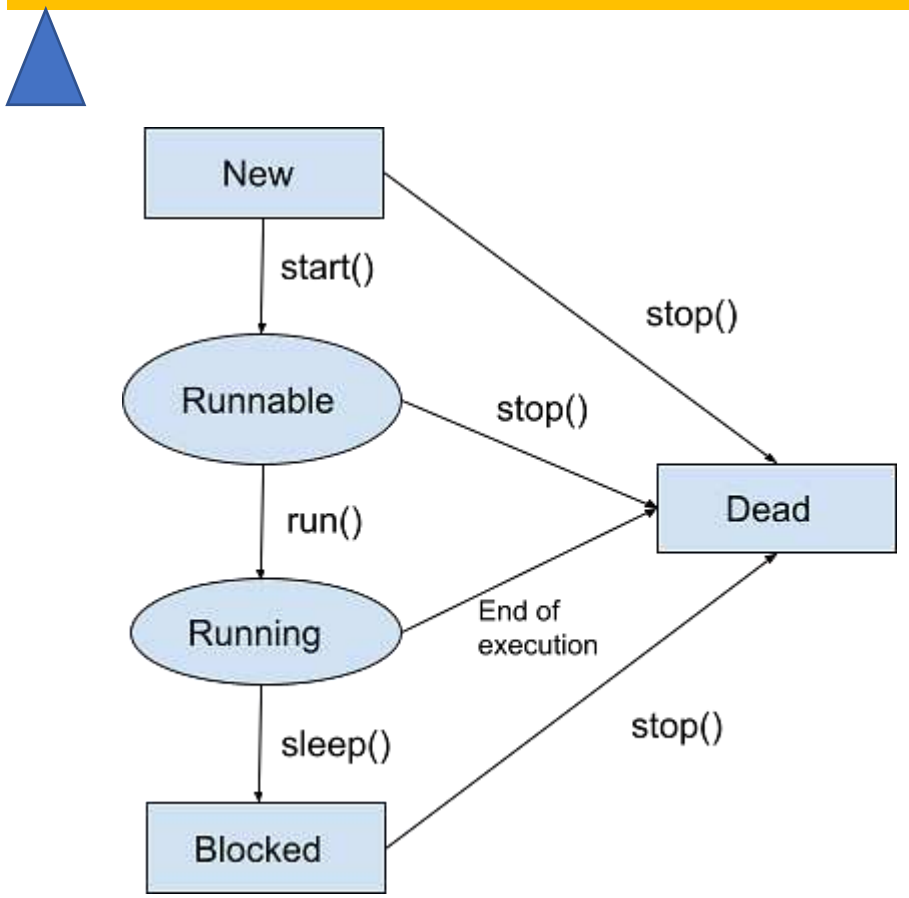

```
class A extends Thread
{
   public void run()
   {
      for(int i=1; i<=5; i++)
        System.out.println("\t From Thread A : i = "+ i);
      System.out.println("Exit From A");
   }
}
class B extends Thread
{
   public void run()
   {
      for(int i=1; i<=5; i++)
        System.out.println("\t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

```
class C extends Thread
{
   public void run()
   {
      for(int i=1; i<=5; i++)
        System.out.println("\t From Thread C : i = "+ i);
      System.out.println("Exit From C");
   }
}
        class  Main
        {
           public static void main (String[] args) {
              new A().start();
              new B().start();
              new C().start();

           }
        }
```

```
From Thread C : i = 1
        From Thread B : i = 1
        From Thread A : i = 1
        From Thread B : i = 2
        From Thread C : i = 2
        From Thread B : i = 3
        From Thread B : i = 4
        From Thread A : i = 2
        From Thread B : i = 5
        From Thread C : i = 3
Exit From B
        From Thread A : i = 3
        From Thread C : i = 4
        From Thread A : i = 4
        From Thread C : i = 5
        From Thread A : i = 5
Exit From C
Exit From A
```

Life Cycle of Thread



The Life Cycle of a Thread in Java refers to the state transformations of a thread that ***begins with its birth*** and ***ends with its death***. When a thread instance is generated and executed by calling the **start() method** of the Thread class, the thread enters the runnable state. When the sleep() or wait() methods of the Thread class are called, the thread enters a non-runnable mode.

Thread returns from non-runnable state to runnable state and starts statement execution. The thread dies when it exits the run() process. In Java, these thread state transformations are referred to as the **Thread life cycle.**

There are basically 4 stages in the lifecycle of a thread, as given below:

1.New
2.Runnable
3.Running
4.Blocked (Non-runnable state)
5.Dead

Contd..

New State

- As we use the Thread class to construct a thread entity, the thread is born and is defined as being in the **New state.** That is, when a thread is created, it enters a new state, but the start() method on the instance has not yet been invoked.

Runnable State

- A thread in the runnable state is prepared to execute the code. When a new thread's start() function is called, it enters a runnable state.
- In the runnable environment, the thread is ready for execution and is awaiting the processor's availability (CPU time). That is, the thread has entered the queue (line) of threads waiting for execution

Contd..

Running State

- Running implies that the processor (CPU) has assigned a time slot to the thread for execution. When a thread from the runnable state is chosen for execution by the thread scheduler, it joins the running state.
- In the running state, the processor allots time to the thread for execution and runs its run procedure. This is the state in which the thread directly executes its operations. Only from the runnable state will a thread enter the running state.

Contd..

Blocked State

- When the thread is alive, i.e., the thread class object persists, but it cannot be selected for execution by the scheduler. It is now inactive.

Dead State

- When a thread's run() function ends the execution of sentences, it automatically dies or enters the dead state. That is, when a thread exits the run() process, it is terminated or killed. When the stop() function is invoked, a thread will also go dead.