Object Oriented Programming

LESSON 01

Multithreading

Outline

1. Introduction
2. Life Cycle of a Thread
3. Thread Priorities
4. Create a Thread using Runnable Interface
5. Create a Thread by extending Thread class

Overview

In this chapter, you are going to learn about

- Know Thread
- Know how to create Thread in Java
- Know how to use Thread
- Know how to use Multiple Thread interaction
- Know how to implement Thread synchronization

1. Multithreading

- Multithreading in Java is a process of executing multiple threads simultaneously.

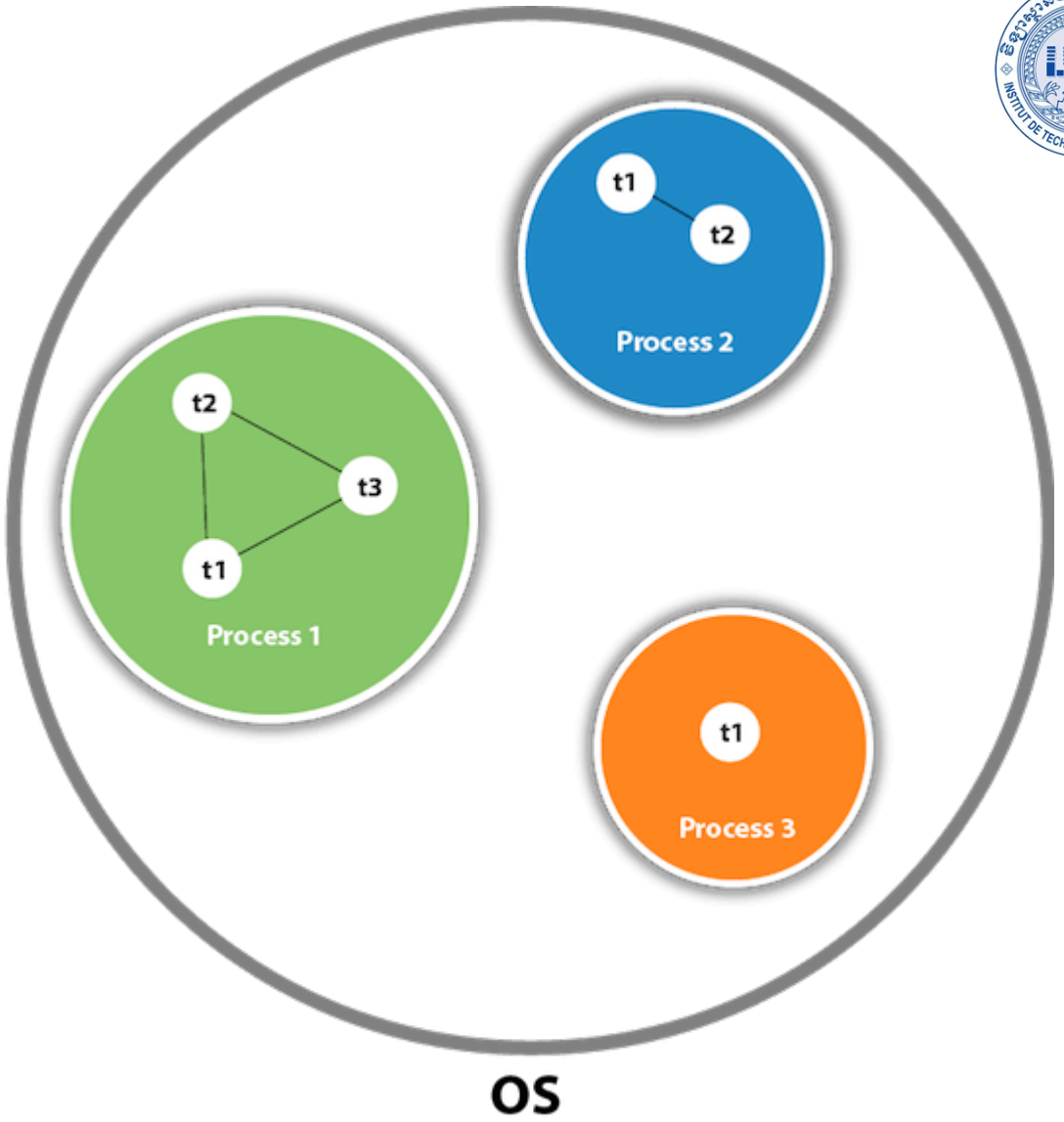
- A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

- However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

- Java is a multi-threaded programming language which means we can develop multi-threaded program using Java.

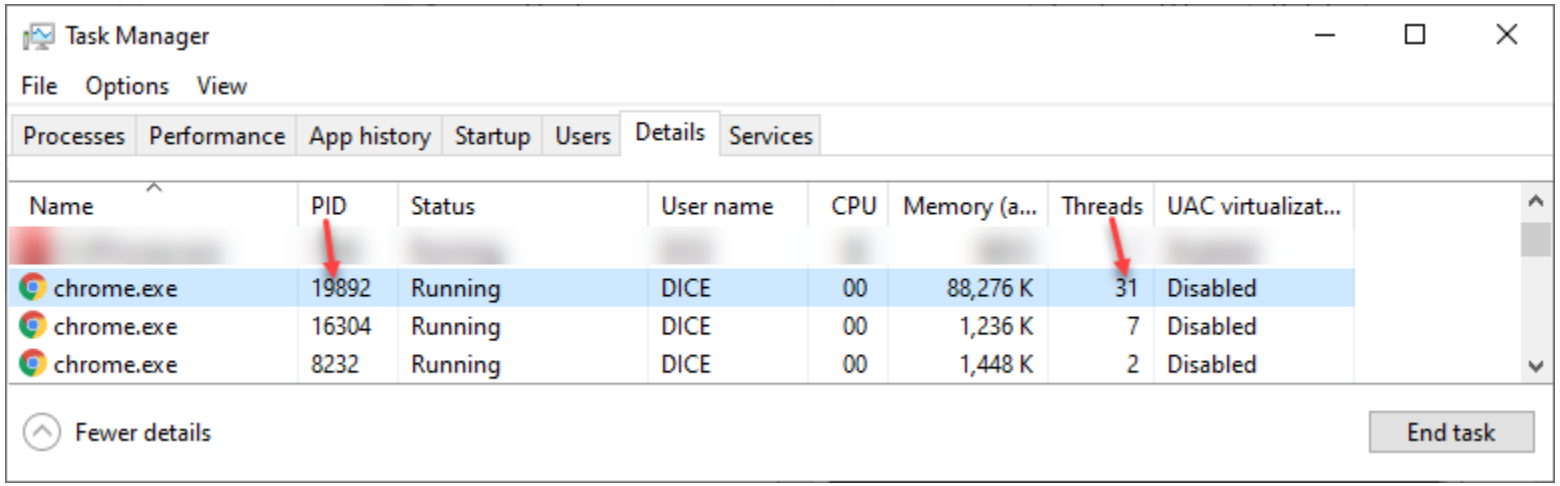
1. Multitasking

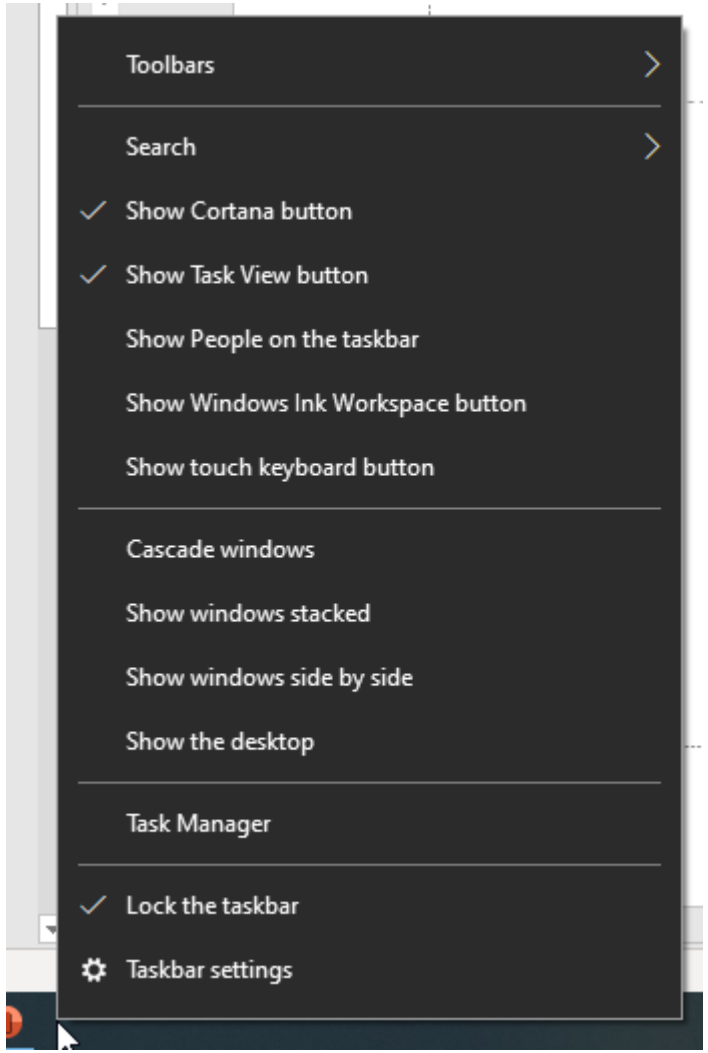
- Multiprocessing and is a Process-based Multitasking. Operating System create a process when running an application. User mays run multiple applications at a time. Each process works independently even if a process is crashed or ended, the OS keeps working as normal. This method made OS more fault-tolerance.



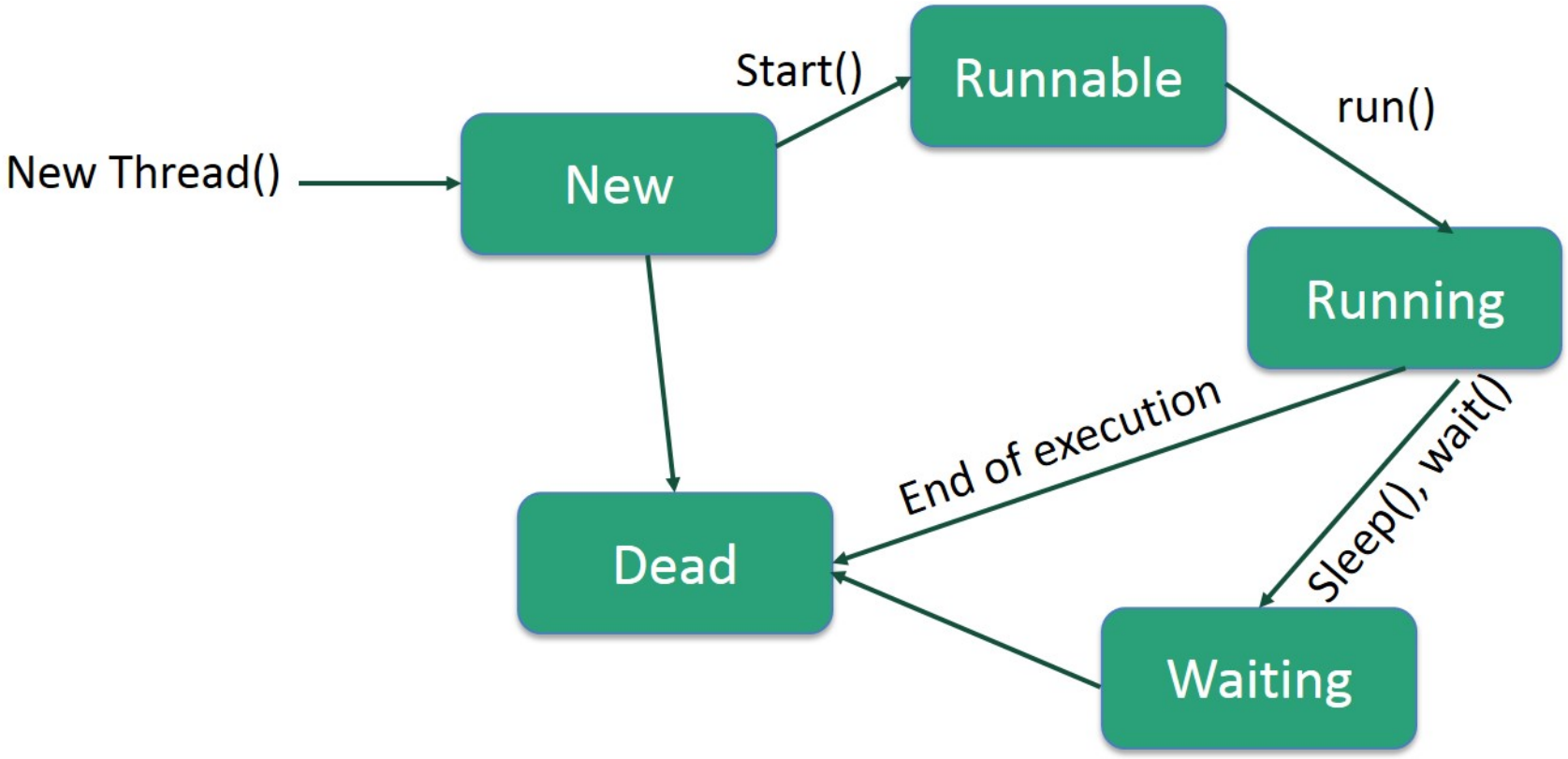
1. Multitasking example

- Task Manager
  (Right click on task bar and choose task manager)

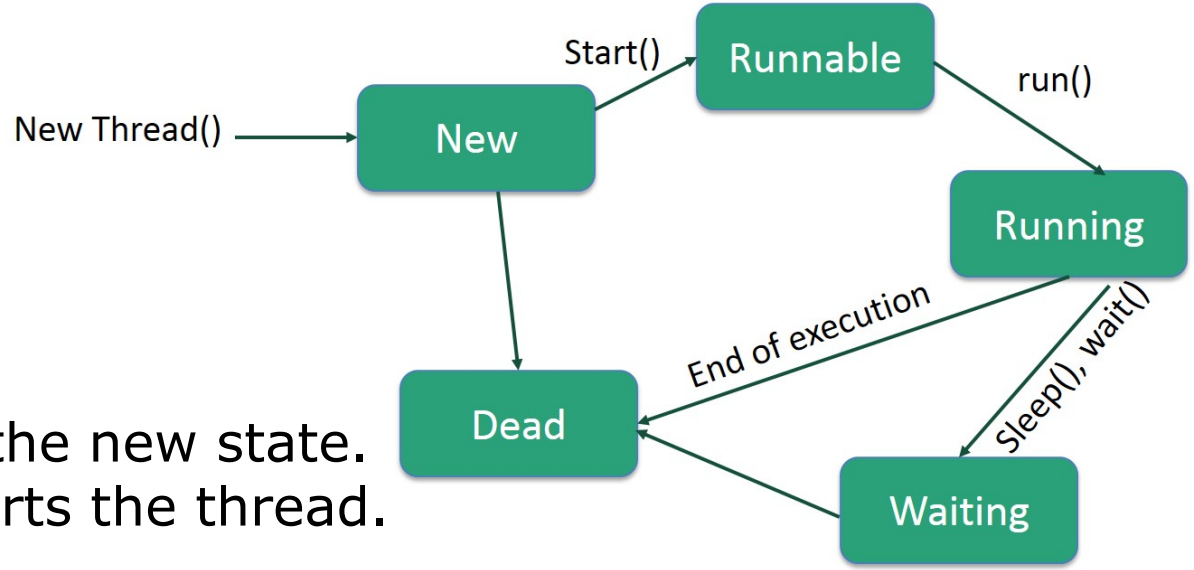




2. Life Cycle of a Thread



2. Life Cycle of a Thread



- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

3. Thread Priorities

- Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.
- Java thread priorities are in the range between **MIN_PRIORITY** (a constant of 1) and **MAX_PRIORITY** (a constant of 10). By default, every thread is given priority **NORM_PRIORITY** (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

4. Create a Thread by Implementing a Runnable Interface

- If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps –
  - Step 1: implement a run() method provided by a **Runnable** interface

    ```
    public void run( )
    ```

  - Step 2: instantiate a **Thread** object using the following constructor

    ```
    Thread(Runnable threadObj, String threadName);
    ```

  - Step 3: start the thread by calling **start()** method

```java
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    RunnableDemo( String name) {
        threadName = name;
        System.out.println("Creating " +  threadName );
    }

    public void run() {
        System.out.println("Running " +  threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " +  threadName + " interrupted.");
        }
        System.out.println("Thread " +  threadName + " exiting.");
    }

    public void start () {
        System.out.println("Starting " +  threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}
```

```java
public class TestThread {

    public static void main(String args[]) {
        RunnableDemo R1 = new RunnableDemo( "Thread-1");
        R1.start();

        RunnableDemo R2 = new RunnableDemo( "Thread-2");
        R2.start();
    }
}
```

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

5. Create a Thread by Extending a Thread Class

- The second way to create a thread is to create a new class that extends **Thread** class
  - Step 1: override **run( )** method available in Thread class
  - Step 2: start the thread by calling **start()** method

```java
class ThreadDemo extends Thread {
    private Thread t;
    private String threadName;

    ThreadDemo( String name) {
        threadName = name;
        System.out.println("Creating " +  threadName );
    }

    public void run() {
        System.out.println("Running " +  threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " + i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " +  threadName + " interrupted.");
        }
        System.out.println("Thread " +  threadName + " exiting.");
    }

    public void start () {
        System.out.println("Starting " +  threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}
```

```java
public class TestThread {

    public static void main(String
args[]) {
        ThreadDemo T1 = new
ThreadDemo( "Thread-1");
        T1.start();

        ThreadDemo T2 = new
ThreadDemo( "Thread-2");
        T2.start();
    }
}
```

Output

```
Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.
```

Reference

- Multithreading in Java – Great Learning
  - https://www.mygreatlearning.com/blog/multithreading-in-java/
- Java – Multithreading
  - https://www.tutorialspoint.com/java/java_multithreading.htm
- Multithreading in Java – Geeks for geeks
  - https://www.geeksforgeeks.org/multithreading-in-java/