Java Threads

What is a thread?

- A thread is a light-weight subprocess.
- For example, in a text editor multiple threads could be data pooling in background, editing, autosaving.
- They share a common area in memory.
- Context switching is relatively fast and easy.
- Threads are independent -> If exception in one thread occurs, other threads are not hampered.

Multi tasking

- Two types of multi tasking

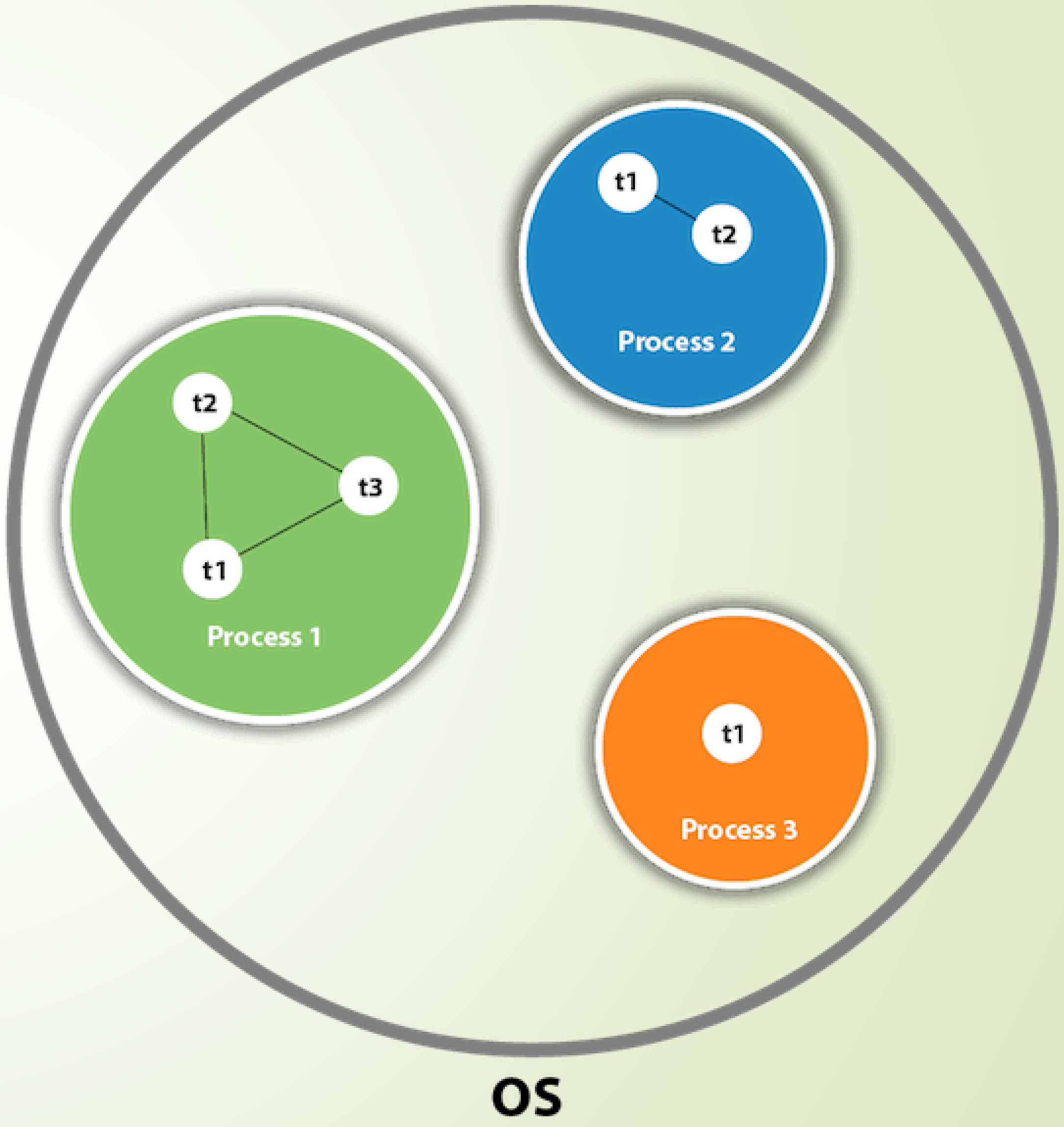
1. Process-based multi tasking
2. Thread based multi tasking

Process-based multi tasking

- Each process has an address space in memory. Each process allocates a separate memory area.
- A process is heavy weight.
- Context switching is heavy as switching from one process to another requires some time for saving and loading registers.

Thread-based multi tasking

- Threads share the same address space.
- It is a light weight sub process.
- Context switching is easy and fast.



Java Thread Class

- Java provides thread class to achieve multi threading.
- It provides constructors and methods and perform operations on a thread.
- We will see some important functions like start(), run(), sleep(), join() etc.

How to create a Java Thread?

- There are two distinct ways to create threads in Java.

1. By extending the Thread class in java
2. By implementing the runnable interface.

Method 1 : Extending the Thread Class

- In order to create a thread using Thread class, we need to extend the Thread class.
- In the inherited class, we need to override the run() method.
- The code that we need to execute as part of the thread must be put inside the run() method.
- In order to execute the thread, we need to call the start() method on an object of the inherited class.
- The start() method will internally call the run() method and execute the thread.

```
Class MyThread extends Thread{
@Override
public void run(){
            //Put your code
            }
}

Class Driver{
            public static void main(Stirng[] args)
            {
                        MyThread t=new MyThread();
                        t.start();
            }
}
```

Method 2 : By implementing the runnable interface

Steps to create threads by Method 2:

1. Create a class and implement the runnable interface using implements keyword.
2. Override the run() method inside the implementer class.
3. Create an object of the implementer class.
4. Instantiate the Thread class and pass the object to the constructor of the Thread class.
5. Call start() on the thread. Start() will internally call run().

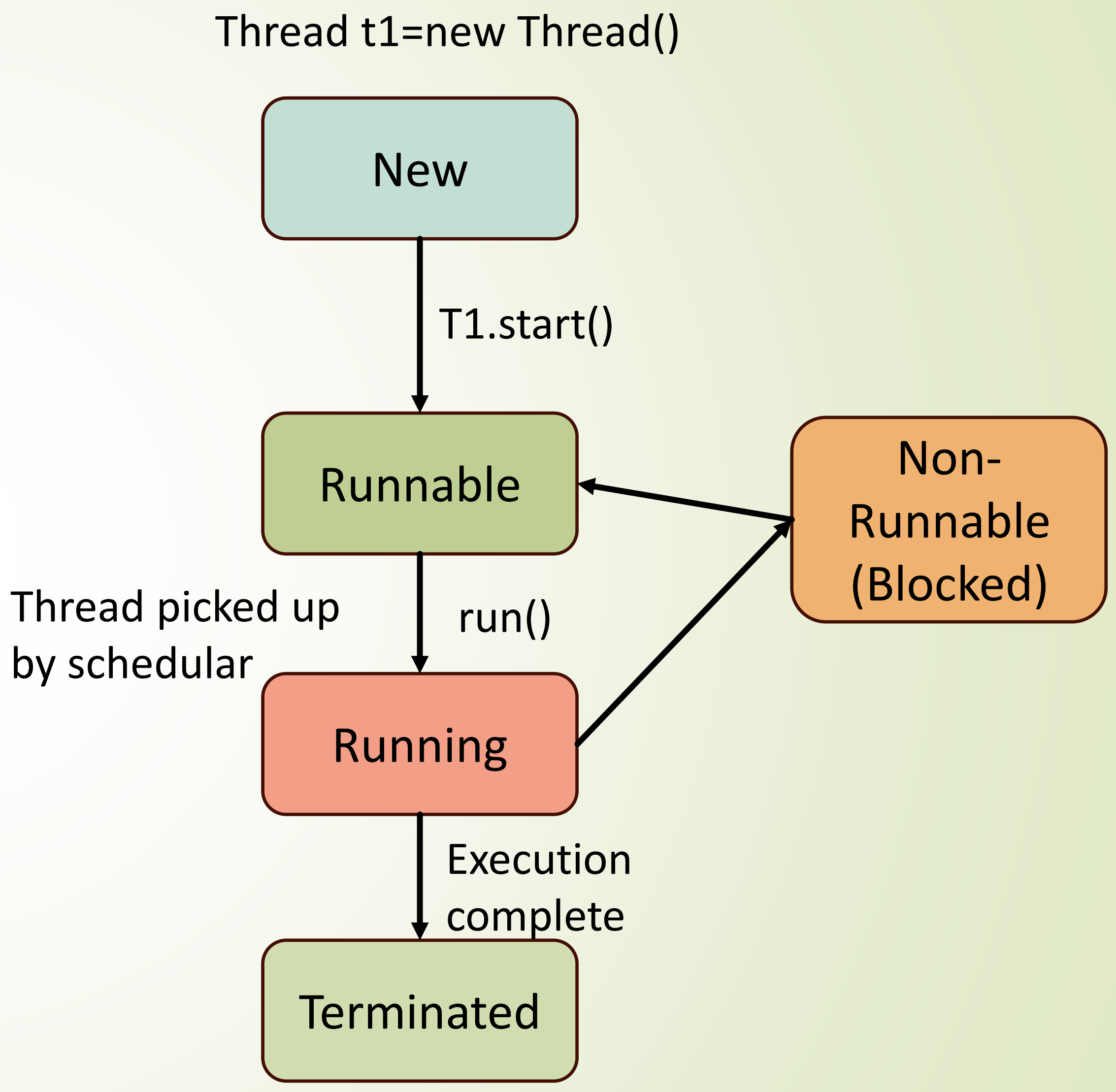
```
classs Thr implements Runnable{
@Override public void run()
        { System.out.println("Thread is running"); }
}

public class ClassName{
        public static void main(String[] args)
        {
                Thr obj1 = new Thr();
                Thread t = new Thread(obj1);
                t.start(); }
        }
}
```

Java Thread Life Cycle

- Java thread life cycle iterates among the 5 different states.

1. New : Instance of thread created which is not yet started by invoking start() method.
2. Runnable : After invocation of start() and before it is selected to be run by the schedular.
3. Running : After the thread schedular has selected it.
4. Non-runnable : Thread alive, but not running.
5. Terminated : run() method completes.



Thread Constructors

- 4 types of Java constructors
  - Thread()
  - Thread (runnable)
  - Thread(String)
  - Thread (runnable, string)

A Few Important Methods

- **Sleep()** : Static method. The thread sleeps for a specific amount of time (Goes to blocked state)
- **join(), join(long mls, int nanos) :** Provided by the java.lang.Thread class that permits one thread to wait until the other thread finish its execution.
- **Yield() :** Static method. Pushes the currently running thread to the runnable state.

Java Synchronization

- It is the capability to control the access of multiple threads to any shared resource.
- Why use Synchronization?
  - To prevent thread interference.
  - To prevent consistency problem
- Two types of synchronization – Process and thread synchronization.
- Thread synchronization – Mutual exclusive and Co-operation.

Mutual Exclusive

- Mutual exclusive helps in threads from interfering with one another while sharing data.
- Ways to achieve ME – Synchronized Method and Synchronized Block

Concept of Lock
Synchronization is built around an entity called lock or monitor. Every object is associated with a lock. If a thread needs consistent access to object’s field, then it has to acquire the lock and later release the lock when its task is complete.

Synchronized Method

- If you declared a method with the keyword **Synchronized**, then it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resources.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes the task.

Synchronized Block

- Perform synchronization on a part of the code.
- Suppose you have a 100 lines of code, out of which you want to synchronize 5 lines, then use a **synchronized** block for the 5 lines.
- If we put all the codes for a function in Synchronized block then it will work same as synchronized method.
- It is used to lock an object for any shared resource.

Syntax :

```
Synchronized (Object reference expression){
        //Your code
}
```

Inter-thread communication

- Co-operation (inter-thread communication) is a mechanism a thread is paused running in its critical section and another thread is allowed to enter the critical section.
- It is implemented by the following functions of the object class.
- Wait(): It causes the current thread to release the lock and wait until either another thread invokes notify() or notifyAll()
- notify() : It wakes up a single thread that is waiting on its object's monitor. If many threads are waiting, one is chosen arbitrarily.
- notifyAll(): Wakes up all the threads that are waiting on the object's monitor

Thank You