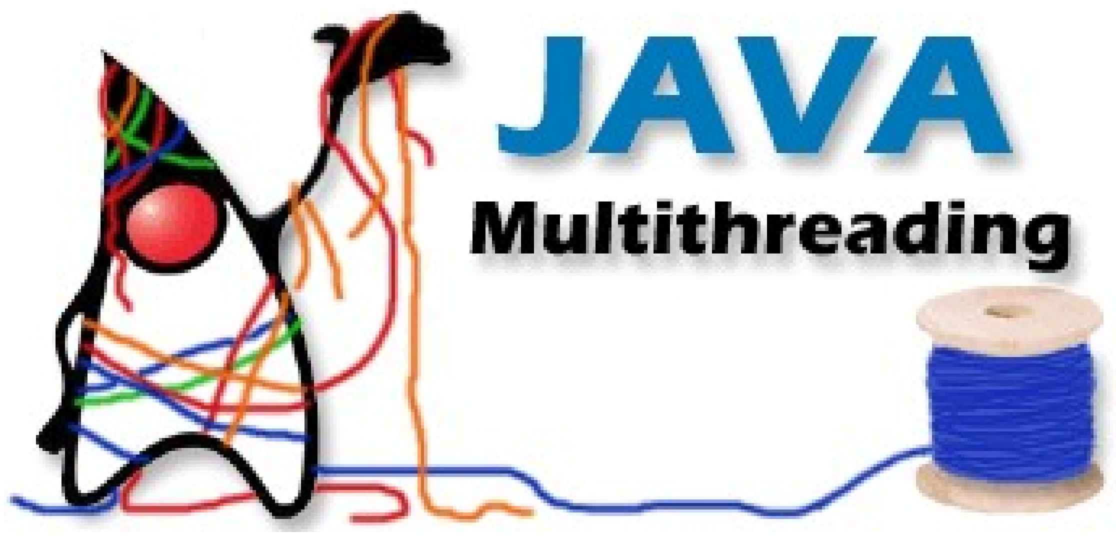


Multithreading in Core JAVA

Multitasking vs Multithreading

Multitasking

A process called multitasking that execute multiple tasks simultaneously.

We can use multitasking to utilize the CPU, for reducing processor ideal time and improving performance.

Multitasking can be achieved by two ways:

 Process-based Multitasking (Multiprocessing)

 Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process have its personal address in the memory. We can say that each process allocates separate memory area.
- Process will be heavy-weight.
- Cost of communication between the processes will be high.
- Switching from one process to any other process require time for saving, loading registers, memory maps, updating lists and various other things.

2) Thread-based Multitasking (Multithreading)

- Threads is shares the same address space.
- Threads will be light-weight than processes.
- Cost of communication between the thread will be lower than processes.

Imp. Note: Thread execut inside the process. We have concept of context-switching between the threads here. Here can be multiple processes inside the Operating System and one process can have more than one threads.

Multithreading

A process is called Multithreading that executes multiple threads simultaneously. Basically, Thread is a lightweight sub-process or we can say a smallest unit of processing. Multiprocessing and multithreading, both can be used to achieve the multitasking but here we use multithreading then multiprocessing because threads share a common memory area and utilize the memory. They do not allocate separate specific memory area and saves memory and context-switching between the various threads takes lesser time than process.

Advantages of Multithreading

- It provides better utilization of system resources, including the CPU.
- It reduces the computation time.
- It improves performance of an application.
- Threads can share the same address space and saves the memory.
- Context switching between threads is usually less expensive then between processes with respect to time.
- Cost of communication between threads is relatively lower than processes.
- Multithreading does not block the user. Reason is that threads are independent and we can perform various operations at same time and save time.
- Threads are independent so it does not affect other threads if exception occur in any one of them.

Thread States or Life cycle of a Thread

The life cycle of any thread in java is fully controlled by JVM (Java Virtual Machine). When any thread will be created, it will go to different states before completing its task than will dead. The different states of any thread are:

1) New
2) Runnable
3) Running
4) Non-Runnable (Blocked)
5) Terminated

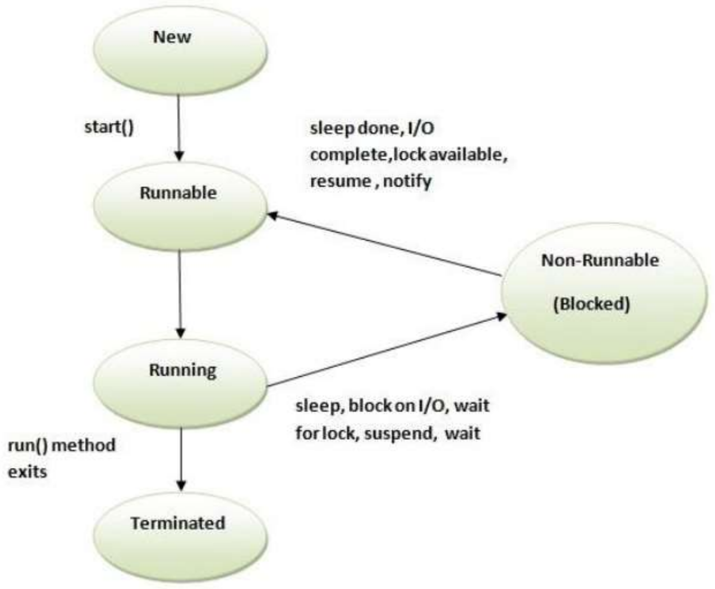
1) **New/Born:** When a thread will be created, it is in new state, in this state thread will not be executed and not sharing time from processor.

2) **Runnable/Ready:** When the start() method is called for the thread object, the thread will be in runnable state. In this state, the thread will be executing and sharing time from the processor.

3) **Running:** If thread scheduler allocates processor to a thread it will go in running state or A thread currently being executed by the CPU is in running state.

4) **Non-Runnable (Blocked):** This is the state when the thread will be still alive, but is currently not eligible to run. A running thread can go to the blocked state due to any of the following conditions.
   - ✓ If wait() or sleep() method is called
   - ✓ If the thread perform I/O operation.
   - ✓ If a blocked thread is unblocked

5) **Terminated:** A thread will be in terminated or dead state when its run() method called. A thread will be dead in two conditions:
   - ✓ If a thread completes its task, it will exit the running state.
   - ✓ If run() method is aborted (due to exception etc.)


New
start()
Runnable
sleep done, I/O complete,lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

CREATION OF THREADS

There are two methods for creating thread in Core-JAVA:

1. By extending Thread class
2. By implementing Runnable interface.

1. **By extending Thread class**

   Class ‘Thread’ in JAVA provides some constructors and methods to create and perform operations on particular thread. Thread class can extends Object class and implements Runnable interface.

   We have some commonly used constructors of Thread class:

   - Thread()
   - Thread(String name)
   - Thread(Runnable r)

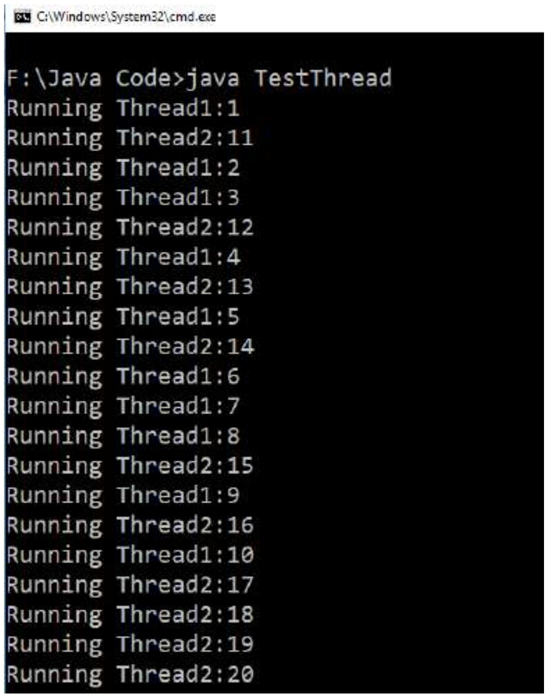
Simple Example of Thread creation by extending thread class

```
class MyThread1 extends Thread
{
  public void run()
  {
    for(int i=1;i<=10;i++)
    {
      System.out.println("Running Thread1:"+i);
      Object Oriented Programming (CSEG2016)
    }
  }
}
class MyThread2 extends Thread
{
  public void run()
  {
    for(int i=11;i<=20;i++)
    {
      System.out.println("Running Thread2:"+i);
    }
  }
}
```

```
class TestThread
{
  public static void main(String arg[])
  {
    MyThread1 mt1=new MyThread1();
    mt1.start();
    MyThread2 mt2=new MyThread2();
    mt2.start();
  }
}
```

Note: In this program both threads are running togather, so we will get mixed output.

Output



C:\Windows\System32\cmd.exe

```
F:\Java Code>java TestThread
Running Thread1:1
Running Thread2:11
Running Thread1:2
Running Thread1:3
Running Thread2:12
Running Thread1:4
Running Thread2:13
Running Thread1:5
Running Thread2:14
Running Thread1:6
Running Thread1:7
Running Thread1:8
Running Thread2:15
Running Thread1:9
Running Thread2:16
Running Thread1:10
Running Thread2:17
Running Thread2:18
Running Thread2:19
Running Thread2:20
```

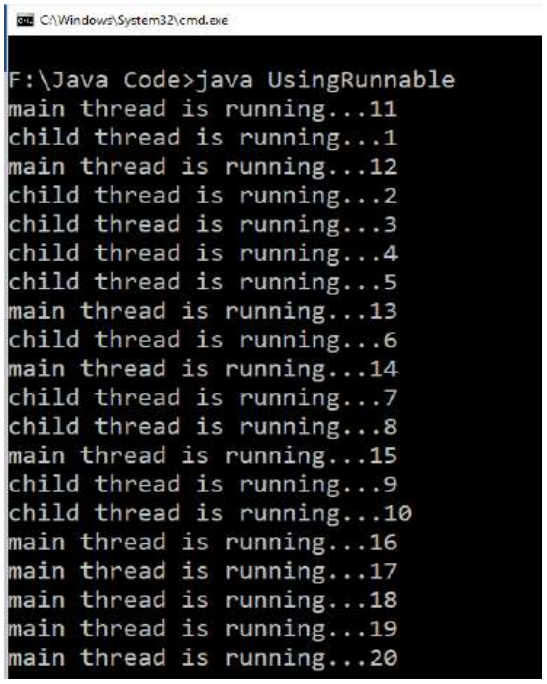
2. By implementing Runnable interface

We can define any thread by implementation of runnable interface. Runnable interface is present in **java.lang package** and it contains only one method run() method.

How we can define a thread through Runnable ?Let's take a simple example by implementing Runnable interface

```java
class UsingRunnable implements Runnable
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        {
          System.out.println("child thread is running..."+i);
        }
    }
    public static void main(String args[])
    {
        UsingRunnable r=new UsingRunnable();
        Thread t =new Thread(r);
        t.start();
        for(int i=11;i<=20;i++)
        {
          System.out.println("main thread is running..."+i);
        }
    }
}
```

Output



C:\Windows\System32\cmd.exe

```
F:\Java Code>java UsingRunnable
main thread is running...11
child thread is running...1
main thread is running...12
child thread is running...2
child thread is running...3
child thread is running...4
child thread is running...5
main thread is running...13
child thread is running...6
main thread is running...14
child thread is running...7
child thread is running...8
main thread is running...15
child thread is running...9
child thread is running...10
main thread is running...16
main thread is running...17
main thread is running...18
main thread is running...19
main thread is running...20
```