**Using Executors API | Fixed Thread Pool**

https://www.youtube.com/watch?v=2jPIsMOfI1o&list=PLq3w0suTCPzbDVkE7RCKrj5MWjneNudfJ&index=10

**2nd class – secondclass.java:**

**import** java.util.concurrent.TimeUnit;

**public** **class** secondclass **implements** Runnable{

**private** **static** **int** *count*=0;

**private** **int** id;

**public** **void** run() {

System.*out*.println("##### <Task-"+id+"> Starting #####");

**for**(**int** i=10;i>0;i--){

System.*out*.println("<Task-"+id+">TICK TICK "+i);

**try**{ TimeUnit.*MILLISECONDS*.sleep((**long**)(Math.*random*()\*1000));

}**catch**(InterruptedException e){

}

}

System.*out*.println("\*\*\*\*\* <Task-"+id+"> DONE \*\*\*\*\*");

}

**public** secondclass(){

**this**.id=++*count*;

}

}

**1st class – apples.java:**

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**public** **class** apples{

**public** **static** **void** main(String[] args){

System.*out*.println("Main thread starts here...");

ExecutorService execService = Executors.*newFixedThreadPool*(3);

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

execService.shutdown();

//Without it the program will still be running and using up memory!

System.*out*.println("Main thread ends here...");

}

}

**Result:**

Main thread starts here...

##### <Task-2> Starting #####

##### <Task-1> Starting #####

Main thread ends here...

##### <Task-3> Starting #####

<Task-1>TICK TICK 10

<Task-2>TICK TICK 10

<Task-3>TICK TICK 10

<Task-2>TICK TICK 9

<Task-2>TICK TICK 8

<Task-2>TICK TICK 7

<Task-1>TICK TICK 9

<Task-3>TICK TICK 9

<Task-3>TICK TICK 8

<Task-3>TICK TICK 7

<Task-3>TICK TICK 6

<Task-2>TICK TICK 6

<Task-1>TICK TICK 8

<Task-1>TICK TICK 7

<Task-3>TICK TICK 5

<Task-2>TICK TICK 5

<Task-2>TICK TICK 4

<Task-2>TICK TICK 3

<Task-1>TICK TICK 6

<Task-3>TICK TICK 4

<Task-2>TICK TICK 2

<Task-2>TICK TICK 1

\*\*\*\*\* <Task-2> DONE \*\*\*\*\*

##### <Task-4> Starting #####

<Task-4>TICK TICK 10

<Task-3>TICK TICK 3

<Task-4>TICK TICK 9

<Task-1>TICK TICK 5

<Task-1>TICK TICK 4

<Task-4>TICK TICK 8

<Task-1>TICK TICK 3

<Task-4>TICK TICK 7

<Task-3>TICK TICK 2

<Task-1>TICK TICK 2

<Task-4>TICK TICK 6

<Task-3>TICK TICK 1

\*\*\*\*\* <Task-3> DONE \*\*\*\*\*

<Task-1>TICK TICK 1

<Task-4>TICK TICK 5

\*\*\*\*\* <Task-1> DONE \*\*\*\*\*

<Task-4>TICK TICK 4

<Task-4>TICK TICK 3

<Task-4>TICK TICK 2

<Task-4>TICK TICK 1

\*\*\*\*\* <Task-4> DONE \*\*\*\*\*

**Important notes:**

1.The programmer introduced 3 thread workers through

“ExecutorService execService = Executors.*newFixedThreadPool*(3);”.

2.When 4 tasks were given, the 3 worker threads started.

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

execService.execute(**new** secondclass());

3.Meanwhile the ExecutorService shutdown right after the 3 threads started.

4. As you can see, the task 4 started after one of the 3 thread workers finished their first task at hand first.

5. Pay close attention to the highlighted code blocks.