Notes - Single Threaded App Example

Serial Threaded vs Multi-threaded -

Multi-threaded applications normally do parallel processing of tasks where as single threaded applications do one task at a time i.e. If there are two tasks such as T1, T2 they are executed in serial order i.e T1 after T2 or T2 after T1, where as multi threading enables us to execute them simultaneously i.e. T1 along with T2. We can choose single threaded applications when parallel processing is not required example use cases such as simple text editor.

Below is a simple example for single threaded application, in this the Task need to print 1500 T's and Main is another task which prints 1500 M's. If we execute this in serial order then it is referred as serial execution.

Serial Execution Example -

```
/*

Some task; here it will print 1500 T's.

*/

class Task {


	public void doTask() {

	for(int i=1; i <= 1500; i++) {

		System.out.print("T");

	}

	}
```

```
}

/* Main */
public class Main {

	public static void main(String[] args) {

	// Print M's
	for(int i=1; i <= 1500; i++) {
		System.out.print("M");
	}

	// Call the task to print T's
	Task t1 = new Task();
	t1.doTask();
	}

}
```

When you run the above example you will see 1500 M's first and then followed by 1500 T's.

Notes - True Parallelism vs Logical Parallelism

True Parallelism vs Logical Parallelism

True parallelism is achieved by assigning tasks to individual CPUs or Processors. This is possible through multicore processors or executing the tasks using multiple CPUs.

If there is one CPU and you want to perform multiple tasks in parallel, then CPU will be shared across those tasks for some stipulated time interval, which stands for interleaved execution, and this way of executing the tasks is known as logical parallelism or psuedo parallelism.

In case of logical parallelism, let’s assume there are two tasks T1 and T2, when executed in parallel and using one CPU, CPU is switched between T1 and T2 i.e. CPU executed T1 for some stipulated time interval and then switched to T2. Once T2's time slice is completed then it is switched back to T1 and starts from where it stops.

Example - (Explained in detail in future lectures.)

In the below example Task is a Thread(explained later), and run is the entry point of the thread execution where it starts printing 1500 T's.

main() runs in Thread i.e. the Main thread which is started by the JVM.

Note In the main method we are not calling doTask directly, instead we are using the start() method of the Thread class, which runs the Task using a separate Thread.

```
class Task extends Thread {

        // Thread execution begins here.
```

```
        public void run() {
        // performs the task i.e. prints 1500 T's
        doTask();
        }

        public void doTask() {
        for(int i=1; i <= 1500; i++) {
                System.out.print("T");
        }
        }
}

public class Main {

        // Runs with in the Main thread started by JVM.
        public static void main(String[] args) {

        Task t1 = new Task();
        // Starts a separate Thread using the
        // the start method of the Thread class.
        t1.start();

        // runs in the Main thread and prints 1500 M's
        for(int i=1; i <= 1500; i++) {
                System.out.print("M");
```

```
        }

        }

}
```

Here main() and Task are run using two separate threads, which means they are executed in parallel (logical parallelism in case of single CPU) and hence you will see output like MMMTTTMMMTTT...