MODULE-3 MultiThreads ASSIGNMENT-

```
//Skill Description:
//Gain hands-on expertise in Java Concurrency and Multithreading by
immersing yourself in the
//captivating narrative of the "Parallel Universe Explorer." Through a
series of tasks, you'll master
//essential concepts, tools, and techniques required for efficient parallel
and reactive programming.

//Problem Statement 2:
//Extend the program to perform a sequential exploration of a timeline
using Sequential Streams.
//Implement a method that simulates sequential exploration tasks.


//Learning Outcomes:
//• Apply Sequential Streams for sequential exploration.
//• Develop skills in simulating sequential exploration tasks.
//This assignment provides a comprehensive journey into Java Concurrency
and Multithreading,
//allowing students to navigate the challenges of the "Parallel Universe
Explorer" and gain practical
//skills in initializing parallel tasks, exploring timelines sequentially
and in parallel, implementing
//reactive communication, and ensuring thread synchronization for shared
resources.


//Code:
package assignments;

import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class ParallelUniverseExplorer  {

	private static final ForkJoinPool pool = new ForkJoinPool();
	private static final List<String> sequentialTimeline =
List.of("one","two", "three", "four", "five");


	public static void main(String[] args) {
		System.out.println("-----task in sequence-----");
		sequence(sequentialTimeline);
		System.out.println("-----task in parallel-----");
		ParallelTask task = new ParallelTask(sequentialTimeline);
		pool.invoke(task);
	}

	static void sequence(List<String> events) {

		events.stream()
			.forEach(event ->System.out.println("task: "+event));
	}
}
```

```
class ParallelTask  extends RecursiveAction{

	private static final long serialVersionUID = 1L;
	private List<String> tasks;
	static final int threshold = 2;
	ParallelTask(List<String> tasks){
		this.tasks=tasks;
	}

	void ParallelPrint(String msg) {
		System.out.println(msg);
		try {
			Thread.sleep(2000);
		}catch(Exception e) {
			System.out.println("exception @ task:" +msg);}
	}

	@Override
	protected void compute() {
		if(tasks.size() <= threshold) {
			tasks.forEach(task->ParallelPrint(task));
		}
		else{
			int mid = tasks.size()/2;
			ParallelTask leftTask = new ParallelTask(tasks.subList(0,
mid));
		  ParallelTask rightTask = new ParallelTask(tasks.subList(mid,
tasks.size()));
		  invokeAll(leftTask, rightTask);
		}
	}

}
```