### **Notes for Candidates:**

* Focus on writing clean, modular, and well-documented code
* Make appropriate use of concurrency mechanisms
* Ensure your solution handles errors gracefully and logs meaningful information
* Proper naming convention for methods and variables

**Code evaluation criteria:**

* Running code (without any errors / warnings)
* Code documentation and structure
* Naming convention of methods / variables
* Using compile time types where applicable
* Concurrency implementation that uses CPU most efficiently without any idle CPU cycles for both network IO and disk IO
* Logging
* Error handling
* Configurability

### **Exercise: Concurrent Task Scheduler**

#### **Problem Statement:**

You are tasked with designing a **Task Scheduler** that can manage and execute tasks concurrently. Each task may take a different amount of time to complete. The scheduler should ensure that no more than **N tasks** run in parallel, and tasks that depend on the completion of other tasks should wait until their dependencies are resolved. Tasks are executed based on priority.

#### **Requirements:**

1. **Task Class:**
   * Create a **Task** class that represents a unit of work.
   * Each task has:
     + A unique task\_id
     + A duration (in seconds, representing the time the task will take to execute)
     + A list of dependencies (other tasks that must complete before this task starts)
     + Priority of task – 1 to 5. 1 is highest priority task and 5 is lowest priority task.
2. **Scheduler Class:**
   * Design a **TaskScheduler** class that:
     + Accepts a list of tasks to execute.
     + Keep an atomic counter of tasks completed.
     + Runs tasks concurrently but limits execution to **N parallel tasks**.
     + Ensures tasks with dependencies do not start until their dependencies are finished.
     + Ensures that tasks with higher priority are executed before the tasks of lower priority.
     + After execution of any tasks updates the atomic counter of tasks completed by one in a thread safe manner.
     + Handles synchronization to avoid race conditions.
3. **Execution:**
   * Simulate task execution by using non-blocking sleep to represent work being done.
   * Print a message when each task starts and completes, including the task ID.
4. **Efficiency:**
   * Ensure that the scheduler minimizes the waiting time by efficiently scheduling tasks.
   * Consider the space and time complexity of managing task dependencies and concurrent execution.

### **Functional Requirements:**

* Implement the task execution such that tasks without dependencies start immediately.
* Tasks with dependencies should wait until all their dependencies complete.
* Tasks with higher priority are executed first.
* The scheduler should support tasks being added dynamically.
* Ensure the solution scales to handle thousands of tasks.

#### **Deliverables:**

1. Source code in any programming language of your choice.
2. A brief explanation of your concurrency implementation.
3. Log output from at least one run of your program

#### **Example Execution:**

Example:

interface Task {

id: string;

processingTime: number;

dependencies: string[];

priority: number;

}

# Defining tasks

tasks = [

{ id: "task1", processingTime: 2, dependencies: [], priority: 1 },

{ id: "task2", processingTime: 1, dependencies: ["task1"], priority: 2 },

{ id: "task3", processingTime: 3, dependencies: ["task1"], priority: 1 },

{ id: "task4", processingTime: 1, dependencies: ["task2","task3"], priority: 3 },

{ id: "task5", processingTime: 2, dependencies: ["task4"], priority: 2 },

{ id: "task6", processingTime: 2, dependencies: ["task5"], priority: 1 },

{ id: "task7", processingTime: 1, dependencies: ["task5"], priority: 3 },

{ id: "task8", processingTime: 2, dependencies: ["task5"], priority: 2 }

];

scheduler = TaskScheduler(max\_concurrent=2)

scheduler.add\_tasks(tasks)

scheduler.run()

Expected output:

Task task1 started. (Priority: 1)

Task task1 completed.

Tasks Completed: 1

Task task3 started. (Priority: 1)

Task task2 started. (Priority: 2)

Task task2 completed.

Tasks Completed: 2

Task task3 completed.

Tasks Completed: 3

Task task4 started. (Priority: 3)

Task task4 completed.

Tasks Completed: 4

Task task5 started. (Priority: 2)

Task task5 completed.

Tasks Completed: 5

Task task6 started. (Priority: 1)

Task task8 started. (Priority: 2)

Task task6 completed.

Tasks Completed: 6

Task task8 completed.

Tasks Completed: 7

Task task7 started. (Priority: 3)

Task task7 completed.

Tasks Completed: 8