기본 파일 구조



### **각 파일 설명 및 초안**

#### **1. Task.java**

* **역할:** 태스크를 모델링하는 클래스입니다. 태스크의 이름, 실행 시간, 우선순위, 상태를 관리합니다.

package com.taskscheduler;

public class Task {

private String name;

private int executionTime;

private int priority;

private TaskState state;

public Task(String name, int executionTime, int priority) {

this.name = name;

this.executionTime = executionTime;

this.priority = priority;

this.state = TaskState.READY;

}

public String getName() {

return name;

}

public int getExecutionTime() {

return executionTime;

}

public int getPriority() {

return priority;

}

public TaskState getState() {

return state;

}

public void setState(TaskState state) {

this.state = state;

}

// 태스크 실행 완료

public void run() {

setState(TaskState.RUNNING);

System.out.println("Executing task: " + name);

try {

Thread.sleep(executionTime \* 1000); // 실행 시간만큼 대기 (초 단위)

} catch (InterruptedException e) {

e.printStackTrace();

}

setState(TaskState.COMPLETED);

System.out.println("Task " + name + " completed.");

}

}

#### **2. TaskState.java**

* **역할:** 태스크의 상태를 나타내는 enum입니다. READY, RUNNING, COMPLETED로 나뉩니다.

package com.taskscheduler;

public enum TaskState {

READY, RUNNING, COMPLETED

}

#### **3. Scheduler.java**

* **역할:** 스케줄러의 인터페이스로, 다양한 스케줄링 알고리즘이 이 인터페이스를 구현하도록 합니다.

package com.taskscheduler;

import java.util.List;

public interface Scheduler {

void schedule(List<Task> tasks);

}

#### **4. FIFOScheduler.java**

* **역할:** FIFO(First-In, First-Out) 방식으로 태스크를 스케줄하는 클래스입니다.

package com.taskscheduler;

import java.util.List;

public class FIFOScheduler implements Scheduler {

@Override

public void schedule(List<Task> tasks) {

System.out.println("Scheduling tasks using FIFO algorithm:");

for (Task task : tasks) {

task.run();

}

}

}

#### **5. RoundRobinScheduler.java**

* **역할:** Round-Robin 방식으로 태스크를 스케줄하는 클래스입니다. 일정 시간 단위로 태스크를 실행합니다.

package com.taskscheduler;

import java.util.List;

public class RoundRobinScheduler implements Scheduler {

private int timeSlice;

public RoundRobinScheduler(int timeSlice) {

this.timeSlice = timeSlice;

}

@Override

public void schedule(List<Task> tasks) {

System.out.println("Scheduling tasks using Round Robin algorithm:");

int time = 0;

while (!tasks.isEmpty()) {

Task task = tasks.remove(0);

System.out.println("Time: " + time + "s, Task " + task.getName() + " running.");

try {

Thread.sleep(timeSlice \* 1000); // 주어진 시간 슬라이스만큼 대기

} catch (InterruptedException e) {

e.printStackTrace();

}

time += timeSlice;

tasks.add(task); // 태스크를 다시 큐에 넣기 (순환)

}

}

}

#### **6. SimulationEngine.java**

* **역할:** 시뮬레이션의 주 엔진입니다. 스케줄러와 태스크를 관리하고 실행을 제어합니다.

package com.taskscheduler;

import java.util.List;

public class SimulationEngine {

private Scheduler scheduler;

public SimulationEngine(Scheduler scheduler) {

this.scheduler = scheduler;

}

public void start(List<Task> tasks) {

scheduler.schedule(tasks);

}

}

#### **7. TaskManager.java**

* **역할:** 태스크 목록을 관리하고, 태스크 생성 및 추가를 담당합니다.

package com.taskscheduler;

import java.util.ArrayList;

import java.util.List;

public class TaskManager {

private List<Task> tasks;

public TaskManager() {

tasks = new ArrayList<>();

}

public void addTask(Task task) {

tasks.add(task);

}

public List<Task> getTasks() {

return tasks;

}

}

#### **8. Main.java**

* **역할:** 프로그램의 진입점이며, 사용자의 입력을 받고 시뮬레이션을 실행하는 역할을 합니다.

package com.taskscheduler;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TaskManager taskManager = new TaskManager();

// 태스크 추가

System.out.print("Enter number of tasks: ");

int n = scanner.nextInt();

scanner.nextLine(); // Consume newline

for (int i = 0; i < n; i++) {

System.out.print("Enter task name: ");

String name = scanner.nextLine();

System.out.print("Enter execution time (in seconds): ");

int executionTime = scanner.nextInt();

System.out.print("Enter priority: ");

int priority = scanner.nextInt();

scanner.nextLine(); // Consume newline

Task task = new Task(name, executionTime, priority);

taskManager.addTask(task);

}

// 스케줄러 선택

System.out.print("Choose scheduler (1: FIFO, 2: Round Robin): ");

int choice = scanner.nextInt();

Scheduler scheduler = (choice == 1) ? new FIFOScheduler() : new RoundRobinScheduler(2); // 예시로 Round Robin의 time slice는 2초

// 시뮬레이션 실행

SimulationEngine engine = new SimulationEngine(scheduler);

engine.start(taskManager.getTasks());

}

}

### **주요 포인트:**

1. **태스크 관리:** Task 클래스는 태스크의 속성을 정의하고, TaskManager 클래스는 태스크 목록을 관리합니다.
2. **스케줄링 알고리즘:** Scheduler 인터페이스와 이를 구현한 FIFOScheduler, RoundRobinScheduler 클래스가 각기 다른 스케줄링 방식을 구현합니다.
3. **시뮬레이션 엔진:** SimulationEngine은 실제 시뮬레이션을 시작하고, 스케줄러를 사용하여 태스크를 실행합니다.
4. **사용자 인터페이스:** Main 클래스는 사용자 입력을 받아 태스크를 생성하고, 어떤 스케줄러를 사용할지 선택합니다.