package astronaut;

import java.time.LocalTime;

import java.time.LocalTime;

import java.util.Scanner;

import java.time.LocalTime;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Astronaut{

public static void main(String[] args) {

ScheduleManager manager = ScheduleManager.getInstance();

TaskFactory factory = new TaskFactory();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\n1. Add Task\n2. Remove Task\n3. View Tasks\n4. View Tasks by Priority\n5. Exit");

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

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

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

String description = scanner.nextLine();

System.out.print("Enter start time (HH:MM): ");

String startTime = scanner.nextLine();

System.out.print("Enter end time (HH:MM): ");

String endTime = scanner.nextLine();

System.out.print("Enter priority (Low, Medium, High): ");

String priority = scanner.nextLine();

Task task = TaskFactory.createTask(description, startTime, endTime, priority);

System.out.println(manager.addTask(task));

break;

case 2:

System.out.print("Enter task description to remove: ");

String removeDescription = scanner.nextLine();

System.out.println(manager.removeTask(removeDescription));

break;

case 3:

System.out.println(manager.viewTasks());

break;

case 4:

System.out.print("Enter priority to filter (Low, Medium, High): ");

String filterPriority = scanner.nextLine();

System.out.println(manager.viewTasksByPriority(filterPriority));

break;

case 5:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please enter a number between 1 and 5.");

break;

}

}

}

}

class Task {

private String description;

private LocalTime startTime;

private LocalTime endTime;

private String priority;

public Task(String description, String startTime, String endTime, String priority) {

this.description = description;

this.startTime = LocalTime.parse(startTime);

this.endTime = LocalTime.parse(endTime);

this.priority = priority;

}

public String getDescription() {

return description;

}

public LocalTime getStartTime() {

return startTime;

}

public LocalTime getEndTime() {

return endTime;

}

public String getPriority() {

return priority;

}

public boolean overlapsWith(Task other) {

return !(this.endTime.isBefore(other.startTime) || this.startTime.isAfter(other.endTime));

}

@Override

public String toString() {

return String.format("%s - %s: %s [%s]", startTime, endTime, description, priority);

}

}

class TaskFactory {

public static Task createTask(String description, String startTime, String endTime, String priority) {

return new Task(description, startTime, endTime, priority);

}

}

class ScheduleManager {

private static ScheduleManager instance;

private List<Task> tasks;

private static final Logger logger = Logger.getLogger(ScheduleManager.class.getName());

private ScheduleManager() {

tasks = new ArrayList<>();

logger.setLevel(Level.INFO);

}

public static synchronized ScheduleManager getInstance() {

if (instance == null) {

instance = new ScheduleManager();

}

return instance;

}

public String addTask(Task task) {

for (Task existingTask : tasks) {

if (task.overlapsWith(existingTask)) {

logger.warning("Task conflict: " + task);

return "Error: Task conflicts with existing tasks.";

}

}

tasks.add(task);

tasks.sort(Comparator.comparing(Task::getStartTime));

logger.info("Task added: " + task);

return "Task added successfully. No conflicts.";

}

public String removeTask(String description) {

for (Task task : tasks) {

if (task.getDescription().equals(description)) {

tasks.remove(task);

logger.info("Task removed: " + task);

return "Task removed successfully.";

}

}

logger.severe("Task not found");

return "Error: Task not found.";

}

public String viewTasks() {

if (tasks.isEmpty()) {

return "No tasks scheduled for the day.";

}

StringBuilder sb = new StringBuilder();

for (Task task : tasks) {

sb.append(task).append("\n");

}

return sb.toString().trim();

}

public String viewTasksByPriority(String priority) {

List<Task> filteredTasks = new ArrayList<>();

for (Task task : tasks) {

if (task.getPriority().equalsIgnoreCase(priority)) {

filteredTasks.add(task);

}

}

if (filteredTasks.isEmpty()) {

return "No tasks with priority " + priority + ".";

}

StringBuilder sb = new StringBuilder();

for (Task task : filteredTasks) {

sb.append(task).append("\n");

}

return sb.toString().trim();

}

}