1. Astronaut Daily Schedule Organizer Programming Exercise

Problem Statement

Design and implement a console-based application that helps astronauts organize their daily schedules. The application should allow users

to add, remove, and view daily tasks. Each task will have a description, start time, end time, and priority level. The intent behind this problem

statement is to evaluate your ability to implement a basic CRUD (Create, Read, Update, Delete) application, manage data efficiently, and

apply best coding practices.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

import java.util.logging.Logger;

// Task class

class Task {

private String description;

private String startTime;

private String endTime;

private String priority;

private boolean completed;

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

this.description = description;

this.startTime = startTime;

this.endTime = endTime;

this.priority = priority;

this.completed = false;

}

public String getDescription() {

return description;

}

public String getStartTime() {

return startTime;

}

public String getEndTime() {

return endTime;

}

public String getPriority() {

return priority;

}

public boolean isCompleted() {

return completed;

}

public void setCompleted(boolean completed) {

this.completed = completed;

}

@Override

public String toString() {

String status = completed ? "Completed" : "Pending";

return startTime + " - " + endTime + ": " + description + " [" + priority + "] - " + status;

}

}

// TaskFactory class

class TaskFactory {

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

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

}

}

// Observer interface

interface Observer {

void update(String message);

}

// ConflictObserver class

class ConflictObserver implements Observer {

@Override

public void update(String message) {

System.out.println("Conflict Observer: " + message);

}

}

// ScheduleManager singleton class

class ScheduleManager {

private static ScheduleManager instance;

private List<Task> tasks;

private static List<Observer> observers = new ArrayList<>();

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

private ScheduleManager() {

tasks = new ArrayList<>();

}

public static ScheduleManager getInstance() {

if (instance == null) {

instance = new ScheduleManager();

}

return instance;

}

public static void addObserver(Observer observer) {

observers.add(observer);

}

public static void notifyObservers(String message) {

for (Observer observer : observers) {

observer.update(message);

}

}

public void addTask(String description, String startTime, String endTime, String priority) {

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

for (Task task : tasks) {

if ((newTask.getStartTime().compareTo(task.getEndTime()) < 0) && (newTask.getEndTime().compareTo(task.getStartTime()) > 0)) {

notifyObservers("Error: Task conflicts with existing task \"" + task.getDescription() + "\".");

return;

}

}

tasks.add(newTask);

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

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

System.out.println("Task added successfully. No conflicts.");

}

public void removeTask(String description) {

for (Task task : tasks) {

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

tasks.remove(task);

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

System.out.println("Task removed successfully.");

return;

}

}

notifyObservers("Error: Task not found.");

}

public void viewTasks() {

if (tasks.isEmpty()) {

System.out.println("No tasks scheduled for the day.");

return;

}

for (Task task : tasks) {

System.out.println(task);

}

}

public void editTask(String oldDescription, String newDescription, String startTime, String endTime, String priority) {

removeTask(oldDescription);

addTask(newDescription, startTime, endTime, priority);

}

public void markTaskCompleted(String description) {

for (Task task : tasks) {

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

task.setCompleted(true);

logger.info("Task marked as completed: " + task);

System.out.println("Task marked as completed.");

return;

}

}

notifyObservers("Error: Task not found.");

}

public void viewTasksByPriority(String priority) {

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

for (Task task : tasks) {

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

tasksWithPriority.add(task);

}

}

if (tasksWithPriority.isEmpty()) {

System.out.println("No tasks with priority " + priority + " found.");

return;

}

for (Task task : tasksWithPriority) {

System.out.println(task);

}

}

}

// Main application

public class AstronautDailyScheduleOrganizer {

public static void main(String[] args) {

ScheduleManager scheduleManager = ScheduleManager.getInstance();

ConflictObserver conflictObserver = new ConflictObserver();

ScheduleManager.addObserver(conflictObserver);

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nAstronaut Daily Schedule Organizer");

System.out.println("1. Add Task");

System.out.println("2. Remove Task");

System.out.println("3. View All Tasks");

System.out.println("4. Edit Task");

System.out.println("5. Mark Task as Completed");

System.out.println("6. View Tasks by Priority");

System.out.println("7. Exit");

System.out.print("Choose an option: ");

String choice = scanner.nextLine();

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 (High/Medium/Low): ");

String priority = scanner.nextLine();

scheduleManager.addTask(description, startTime, endTime, priority);

break;

case "2":

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

description = scanner.nextLine();

scheduleManager.removeTask(description);

break;

case "3":

scheduleManager.viewTasks();

break;

case "4":

System.out.print("Enter the description of the task to edit: ");

String oldDescription = scanner.nextLine();

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

String newDescription = scanner.nextLine();

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

startTime = scanner.nextLine();

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

endTime = scanner.nextLine();

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

priority = scanner.nextLine();

scheduleManager.editTask(oldDescription, newDescription, startTime, endTime, priority);

break;

case "5":

System.out.print("Enter task description to mark as completed: ");

description = scanner.nextLine();

scheduleManager.markTaskCompleted(description);

break;

case "6":

System.out.print("Enter priority level to view (High/Medium/Low): ");

priority = scanner.nextLine();

scheduleManager.viewTasksByPriority(priority);

break;

case "7":

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

}

// Command Interface

interface TaskCommand {

void execute();

}

// Concrete Command: Add Task

class AddTaskCommand implements TaskCommand {

private ScheduleManager scheduleManager;

private String description;

private String startTime;

private String endTime;

private String priority;

public AddTaskCommand(ScheduleManager scheduleManager, String description, String startTime, String endTime, String priority) {

this.scheduleManager = scheduleManager;

this.description = description;

this.startTime = startTime;

this.endTime = endTime;

this.priority = priority;

}

@Override

public void execute() {

scheduleManager.addTask(description, startTime, endTime, priority);

}

}

// Concrete Command: Remove Task

class RemoveTaskCommand implements TaskCommand {

private ScheduleManager scheduleManager;

private String description;

public RemoveTaskCommand(ScheduleManager scheduleManager, String description) {

this.scheduleManager = scheduleManager;

this.description = description;

}

@Override

public void execute() {

scheduleManager.removeTask(description);

}

}

// Concrete Command: Mark Task as Completed

class MarkTaskCompletedCommand implements TaskCommand {

private ScheduleManager scheduleManager;

private String description;

public MarkTaskCompletedCommand(ScheduleManager scheduleManager, String description) {

this.scheduleManager = scheduleManager;

this.description = description;

}

@Override

public void execute() {

scheduleManager.markTaskCompleted(description);

}

}

// Updated ScheduleManager to act as Invoker

class ScheduleManager {

private static ScheduleManager instance;

private List<Task> tasks;

private static List<Observer> observers = new ArrayList<>();

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

private ScheduleManager() {

tasks = new ArrayList<>();

}

public static ScheduleManager getInstance() {

if (instance == null) {

instance = new ScheduleManager();

}

return instance;

}

public static void addObserver(Observer observer) {

observers.add(observer);

}

public static void notifyObservers(String message) {

for (Observer observer : observers) {

observer.update(message);

}

}

public void addTask(String description, String startTime, String endTime, String priority) {

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

for (Task task : tasks) {

if ((newTask.getStartTime().compareTo(task.getEndTime()) < 0) && (newTask.getEndTime().compareTo(task.getStartTime()) > 0)) {

notifyObservers("Error: Task conflicts with existing task \"" + task.getDescription() + "\".");

return;

}

}

tasks.add(newTask);

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

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

System.out.println("Task added successfully. No conflicts.");

}

public void removeTask(String description) {

for (Task task : tasks) {

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

tasks.remove(task);

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

System.out.println("Task removed successfully.");

return;

}

}

notifyObservers("Error: Task not found.");

}

public void markTaskCompleted(String description) {

for (Task task : tasks) {

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

task.setCompleted(true);

logger.info("Task marked as completed: " + task);

System.out.println("Task marked as completed.");

return;

}

}

notifyObservers("Error: Task not found.");

}

// Other methods remain the same

}

// Main application demonstrating the use of Command pattern

public class AstronautDailyScheduleOrganizer {

public static void main(String[] args) {

ScheduleManager scheduleManager = ScheduleManager.getInstance();

ConflictObserver conflictObserver = new ConflictObserver();

ScheduleManager.addObserver(conflictObserver);

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nAstronaut Daily Schedule Organizer");

System.out.println("1. Add Task");

System.out.println("2. Remove Task");

System.out.println("3. View All Tasks");

System.out.println("4. Edit Task");

System.out.println("5. Mark Task as Completed");

System.out.println("6. View Tasks by Priority");

System.out.println("7. Exit");

System.out.print("Choose an option: ");

String choice = scanner.nextLine();

TaskCommand command = null;

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 (High/Medium/Low): ");

String priority = scanner.nextLine();

command = new AddTaskCommand(scheduleManager, description, startTime, endTime, priority);

break;

case "2":

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

description = scanner.nextLine();

command = new RemoveTaskCommand(scheduleManager, description);

break;

case "5":

System.out.print("Enter task description to mark as completed: ");

description = scanner.nextLine();

command = new MarkTaskCompletedCommand(scheduleManager, description);

break;

case "3":

scheduleManager.viewTasks();

break;

case "4":

System.out.print("Enter the description of the task to edit: ");

String oldDescription = scanner.nextLine();

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

String newDescription = scanner.nextLine();

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

startTime = scanner.nextLine();

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

endTime = scanner.nextLine();

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

priority = scanner.nextLine();

scheduleManager.editTask(oldDescription, newDescription, startTime, endTime, priority);

break;

case "6":

System.out.print("Enter priority level to view (High/Medium/Low): ");

priority = scanner.nextLine();

scheduleManager.viewTasksByPriority(priority);

break;

case "7":

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

}

if (command != null) {

command.execute();

}

}

}

}