Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи № 8 з дисципліни

«Основи програмування. Частина 2. Методології програмування»

„ ВІДНОШЕННЯ МІЖ КЛАСАМИ ТА ОБ'ЄКТАМИ”

Виконав(ла) ІП-42 Кравченко Роман Олександрович

Перевірив Куценко Микита Олександрович

Київ 2025

Лабораторна робота №8

Мета лабораторної роботи – дослідити типи відношень між класами та об’єктами в ООП, навчитися проектувати об’єктно-орієнтовану модель предметної галузі.

Завдання

1. Вивчити типи відношень між класами в ООП.
2. Спроектувати об’єктно-орієнтовану модель предметної галузі згідно з варіантом, визначивши необхідні для цього класи та їх структуру**.**
3. Вимоги до проектування:
   - розробити не менше 6 типів даних;
   - застосувати всі базові принципи ООП;
   - застосувати всі види відношень;
   - застосувати обробку виключень, де це є необхідним;
   - дотримуватись єдиної конвенції найменувань та принципів написання "чистого" коду;
   - код дозволяється коментувати лише xml-коментарями.
4. Написати програму, в якій реалізувати попередньо спроектовану об’єктно-орієнтовану модель.
5. Програмний інтерфейс, наприклад, введення\виведення з консолі, реалізовувати окремим проектом. Код програмного інтерфейсу має бути простим (демонструється використання класів предметної галузі шляхом створення об’єктів та їх застосування, відсутня перевірка коректності вводу, введення з консолі мінімальне або відсутнє взагалі).

Варіант

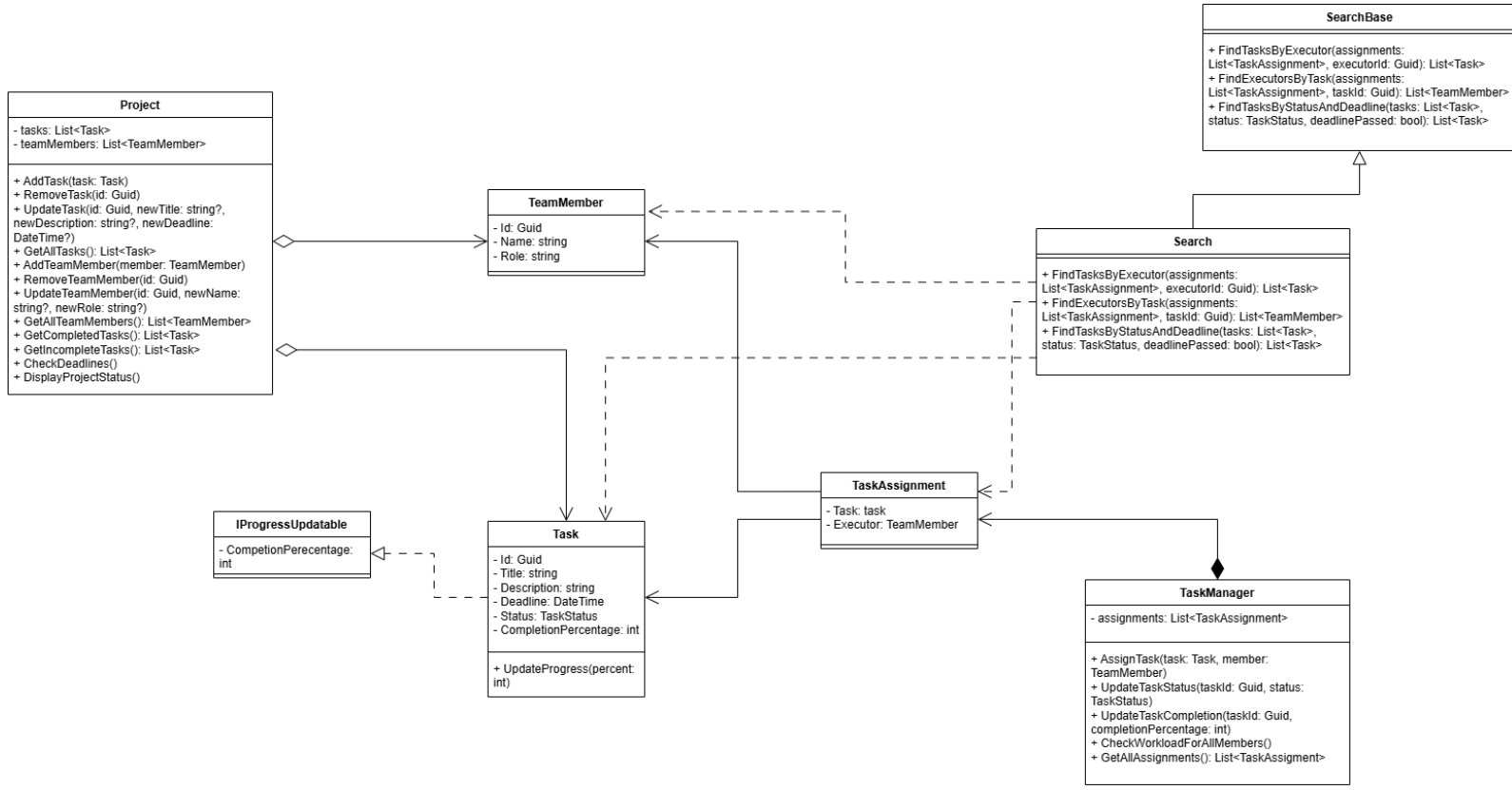
Варіант 14

Планувальник завдань: розподіл та контроль за виконанням завдань членами команди проекту

Функціональні вимоги до програмного забезпечення

1. Управління членами команди
   1.1. Можливість додавати виконавця
   1.2. Можливість видаляти виконавця
   1.3. Можливість змінити дані про виконавця
   1.4. Можливість перегляду списку всіх членів команди
2. Управління завданнями
   2.1. Можливість додавати завдання
   2.2. Можливість видаляти завдання
   2.3. Можливість змінювати дані завдання
   2.4. Можливість перегляду списку завдань
   2.5. Можливість перегляду завдань виконаних\невиконаних
3. Управління розподілом та виконанням завдань
   3.1. Можливість розподілити завдання між виконавцями
   3.2. Можливість вказати ступінь виконання, статус виконано\не виконано
   3.3. Можливість перевірки терміну виконання (триває\закінчився)
   3.4. Можливість перевірки завантаженості виконавців
   3.5. Можливість отримання стану виконання проекту
4. Пошук
   4.1. Можливість пошуку виконавця та його завдань
   4.2. Можливість пошуку виконавців певного завдання
   4.3. Можливість пошуку виконаних\невиконаних завдань, термін яких сплив\триває

Діаграма класів



Вихідний код

TaskScheduler:

```
public enum TaskStatus
{
    NotStarted,
    InProgress,
    Completed
}

public class TeamMember
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Role { get; set; }
```

```
    public TeamMember(string name, string role)
    {
        Id = Guid.NewGuid();
        Name = name;
        Role = role;
    }
}

public interface IProgressUpdatable
{
    int CompletionPercentage { get; }
}

public class Task : IProgressUpdatable
{
    public Guid Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime Deadline { get; set; }
    public TaskStatus Status { get; set; }
    public int CompletionPercentage { get; private set; }

    public Task(string title, string description, DateTime deadline)
    {
        Id = Guid.NewGuid();
        Title = title;
        Description = description;
        Deadline = deadline;
```

```
        Status = TaskStatus.NotStarted;
        CompletionPercentage = 0;
    }

    internal void UpdateProgress(int percent)
    {
        CompletionPercentage = Math.Clamp(percent, 0, 100);
        Status = percent == 100 ? TaskStatus.Completed :
                 percent > 0 ? TaskStatus.InProgress :
                 TaskStatus.NotStarted;
    }
}

public class TaskAssignment
{
    public Task Task { get; set; }
    public TeamMember Executor { get; set; }
}

public class Project
{
    private List<Task> tasks = new();
    private List<TeamMember> teamMembers = new();

    public void AddTask(Task task) => tasks.Add(task);
    public void RemoveTask(Guid id) => tasks.RemoveAll(t => t.Id == id);
    public void UpdateTask(Guid id, string? newTitle = null, string?
newDescription = null, DateTime? newDeadline = null)
    {
```

```
        var task = tasks.FirstOrDefault(t => t.Id == id);
        if (task == null) throw new Exception("Task not found.");

        if (newTitle != null) task.Title = newTitle;
        if (newDescription != null) task.Description = newDescription;
        if (newDeadline != null) task.Deadline = (DateTime)newDeadline;
    }

    public List<Task> GetAllTasks() => tasks;

    public void AddTeamMember(TeamMember member) =>
teamMembers.Add(member);
    public void RemoveTeamMember(Guid id) => teamMembers.RemoveAll(m
=> m.Id == id);
    public void UpdateTeamMember(Guid id, string? newName = null, string?
newRole = null)
    {
        var member = teamMembers.FirstOrDefault(m => m.Id == id);
        if (member == null) throw new Exception("Team member not found.");

        if (newName != null) member.Name = newName;
        if (newRole != null) member.Role = newRole;
    }

    public List<TeamMember> GetAllTeamMembers() => teamMembers;

    public List<Task> GetCompletedTasks() => tasks.Where(t => t.Status ==
TaskStatus.Completed).ToList();
```

```
    public List<Task> GetIncompleteTasks() => tasks.Where(t => t.Status !=
TaskStatus.Completed).ToList();

    public void CheckDeadlines()
    {
        foreach (var task in tasks)
        {
            string status = task.Deadline < DateTime.Now ? "deadline passed!" :
"within deadline.";
            Console.WriteLine($"Task '{task.Title}' is {status}");
        }
    }

    public void DisplayProjectStatus()
    {
        foreach (var task in tasks)
        {
            Console.WriteLine($"- {task.Title}: {task.Status}
({task.CompletionPercentage}%)");
        }
    }
}

public class TaskManager()
{
    private List<TaskAssignment> assignments = new();

    public void AssignTask(Task task, TeamMember member)
    {
```

```
        assignments.Add(new TaskAssignment { Task = task, Executor = member
});
        task.Status = TaskStatus.InProgress;
    }

    public void UpdateTaskStatus(Guid taskId, TaskStatus status)
    {
        var assignment = assignments.FirstOrDefault(a => a.Task.Id == taskId);
        if (assignment == null) throw new Exception("Task not assigned.");

        assignment.Task.Status = status;
        if (status == TaskStatus.Completed)
assignment.Task.UpdateProgress(100);
        if (status == TaskStatus.NotStarted) assignment.Task.UpdateProgress(0);
    }

    public void UpdateTaskCompletion(Guid taskId, int completionPercentage)
    {
        var assignment = assignments.FirstOrDefault(a => a.Task.Id == taskId);
        if (assignment == null) throw new Exception("Task not assigned.");

        assignment.Task.UpdateProgress(completionPercentage);
    }

    public void CheckWorkloadForAllMembers()
    {
        var grouped = assignments.GroupBy(a => a.Executor);

        foreach (var group in grouped)
```

```
        {
            Console.WriteLine($"{group.Key.Name} ({group.Key.Role}) - Tasks:
{group.Count()}");
        }
    }

    public List<TaskAssignment> GetAllAssignments() => assignments;
}

public abstract class SearchBase
{
    public abstract List<Task> FindTasksByExecutor(List<TaskAssignment>
assignments, Guid executorId);
    public abstract List<TeamMember>
FindExecutorsByTask(List<TaskAssignment> assignments, Guid taskId);
    public abstract List<Task> FindTasksByStatusAndDeadline(List<Task>
tasks, TaskStatus status, bool deadlinePassed);
}

public class Search : SearchBase
{
    public override List<Task> FindTasksByExecutor(List<TaskAssignment>
assignments, Guid executorId)
    {
        return assignments
            .Where(a => a.Executor.Id == executorId)
            .Select(a => a.Task)
            .ToList();
    }
```

```
    public override List<TeamMember>
FindExecutorsByTask(List<TaskAssignment> assignments, Guid taskId)
    {
        return assignments
            .Where(a => a.Task.Id == taskId)
            .Select(a => a.Executor)
            .ToList();
    }

    public override List<Task> FindTasksByStatusAndDeadline(List<Task>
tasks, TaskStatus status, bool deadlinePassed)
    {
        return tasks
            .Where(t => t.Status == status && (deadlinePassed ? t.Deadline <
DateTime.Now : t.Deadline >= DateTime.Now))
            .ToList();
    }
}
```

Program:

```
static void Main(string[] args)
{
    //Створюємо проєкт
    Project project = new Project();
    TaskManager manager = new TaskManager();
    Search search = new Search();

    //Створюємо членів команди
```

```
    TeamMember mykyta = new TeamMember("Mykyta", "Backend
Developer");
    TeamMember olena = new TeamMember("Olena", "Team Lead");
    TeamMember serhii = new TeamMember("Serhii", "Full Stack Developer");
    TeamMember anhelina = new TeamMember("Anhelina", "Designer");
    TeamMember bohdan = new TeamMember("Bohdan", "Backend
Developer");
    TeamMember roman = new TeamMember("Roman", "Backend Developer");



    project.AddTeamMember(mykyta);
    project.AddTeamMember(olena);
    project.AddTeamMember(serhii);
    project.AddTeamMember(anhelina);
    project.AddTeamMember(bohdan);
    project.AddTeamMember(roman);

    //Тепер проводемо певні маніпуляції над ними
    project.UpdateTeamMember(mykyta.Id, newRole: "Full Stack Developer");
    project.RemoveTeamMember(olena.Id);

    WriteLine("--- Our team ---");
    List<TeamMember> team = project.GetAllTeamMembers();

    foreach (var teamMember in team)
    {
        WriteLine($"{teamMember.Name} as {teamMember.Role}");
    }
    WriteLine();
```

```
    //Тепер створимо завдання
    Task task1 = new Task("Implement login", "Login with email",
DateTime.Now.AddDays(3));
    Task task2 = new Task("Create UI mockup", "Design login screen",
DateTime.Now.AddDays(1));
    Task task3 = new Task("Fix bug #45", "Resolve critical issue",
DateTime.Now.AddDays(5));
    Task task4 = new Task("Create homepage wireframe", "Design a low-fidelity
wireframe for the homepage layout", DateTime.Now.AddDays(5));
    Task task5 = new Task("Fix bug #23", "Resolve critical issue",
DateTime.Now.AddDays(-4));

    project.AddTask(task1);
    project.AddTask(task2);
    project.AddTask(task3);
    project.AddTask(task4);
    project.AddTask(task5);

    //Тепер проведемо маніпуляції і над завданнями
    project.UpdateTask(task2.Id, newDeadline: DateTime.Now.AddDays(1));
    project.RemoveTask(task4.Id);

    //Призначимо наші завдання членам команди
    manager.AssignTask(task1, mykyta);
    manager.AssignTask(task3, mykyta);
    manager.AssignTask(task2, anhelina);
    manager.AssignTask(task3, serhii);
```

```
manager.AssignTask(task4, anhelina);
manager.AssignTask(task1 , roman);

//Встановимо статуси завданням
manager.UpdateTaskStatus(task3.Id, TaskStatus.Completed);
manager.UpdateTaskCompletion(task1.Id, completionPercentage: 60);

//Виведемо інформацію про завдання
WriteLine("--- Tasks ---");
List<Task> tasks = project.GetAllTasks();

foreach (var task in tasks)
{
    WriteLine($"{task.Title}: {task.Description}");
}

WriteLine("\nCompleted Tasks:");
List<Task> completedTasks = project.GetCompletedTasks();
foreach (var task in completedTasks)
{
    WriteLine($"{task.Title}: {task.Description} ({task.Status})");
}

WriteLine("\nIncomplete Tasks:");
List<Task> incompleteTasks = project.GetIncompleteTasks();
foreach (var task in incompleteTasks)
{
    WriteLine($"{task.Title}: {task.Description} ({task.Status})");
}
```

```
    WriteLine();

    //Перевіримо термін виконання
    WriteLine("--- Deadlines ---");
    project.CheckDeadlines();
    WriteLine();

    //Перевіримо завантаженість виконавців
    WriteLine("--- Workload ---");
    manager.CheckWorkloadForAllMembers();
    WriteLine();

    //Перевіримо стан виконання проекту
    WriteLine("--- Project status ---");
    project.DisplayProjectStatus();
    WriteLine();

    //Перевіримо функціонал пошуку
    WriteLine("--- Search ---");

    WriteLine("Tasks taken by 'Mykyta':");
    var tasksByMykyta =
search.FindTasksByExecutor(manager.GetAllAssignments(), mykyta.Id);
    foreach (var task in tasksByMykyta)
    {
        WriteLine($"- {task.Title}");
    }
    WriteLine();
```

```
    WriteLine($"Executors of the task '{task2.Title}':");
    var executorsOfTask1 =
search.FindExecutorsByTask(manager.GetAllAssignments(), task1.Id);
    foreach (var task in executorsOfTask1)
    {
        WriteLine($"- {task.Name} ({task.Role})");
    }
    WriteLine();

    WriteLine("Uncompleted tasks with expired deadlines:");
    var overdueNotCompleted =
search.FindTasksByStatusAndDeadline(project.GetAllTasks(),
TaskStatus.NotStarted, deadlinePassed: true);
    foreach (var task in overdueNotCompleted)
    {
        WriteLine($"- {task.Title}");
    }
    WriteLine();

    WriteLine("Completed tasks that have not yet expired:");
    var completedStillInTime =
search.FindTasksByStatusAndDeadline(project.GetAllTasks(),
TaskStatus.Completed, deadlinePassed: false);
    foreach (var task in completedStillInTime)
    {
        WriteLine($"- {task.Title}");
    }
}
```

Висновок

Отже, нами було створено власну ООП модель планувальника завдань. Увесь її функціонал було протестовано, тим самим довівши коректність роботи.