МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Лабораторна робота № 2**

з дисципліни « Об’єктно орієнтоване програмування »

*назва дисципліни*

на тему: « КЛАСИ, ОБ’ЄКТИ »

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: доц. Вдовітченко О.В.

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

# ПОСТАНОВКА ЗАДАЧІ

**Завдання**

На основі отриманого на лекції 2 теоретичного матеріалу скорегувати програму для лабораторної роботи № 1 наступним чином:

1. Всі поля (характеристики) класу повинні бути інкапсульовані за допомогою модифікатора доступу private.
2. Для кожного private-поля, яке необхідне для зовнішньої взаємодії, в класі мають бути додані відповідні public-властивості (public- properties)\*.

\*якщо предметна область вимагає, то секція set властивості обов’язково повинна мати відповідну перевірку.

1. До класу додати хоча б одну автовластивість\*\*.

\*\*передбачте значення за замовчуванням.

1. До класу додати хоча б одну обчислювальну властивість, яка не буде пов’язана з будь-яким полем класу і буде використовувати інші властивості та/або методи.
2. Хоча б одна властивість повинна мати різній рівень доступу для секцій get і set\*\*\*.

\*\*\*пам’ятаємо, що модифікатор доступу секції має бути більш обмежувальним, ніж модифікатор самої властивості.

1. Встановлення та зчитування значень полів в основній програмі реалізувати через додані у клас властивості.
2. Меню в програмі залишається з лабораторної роботи № 1:
3. додати об’єкт
4. вивести на екран об’єкти
5. знайти об’єкт
6. видалити об’єкт
7. демонстрація поведінки
8. вийти з програми
9. Детально протестувати програму. Мають бути протестовані всі пункти меню. Проводимо тестування не тільки на коректних значеннях, а також не забуваємо перевірити і некоректні значення, які може ввести користувач.
10. Оформити звіт:

* Титульний аркуш
* Завдання
* Короткий опис класу (опис характеристик та поведінки з точки зору замовника, не забути додати нову інформацію щодо класу)
* Програмна реалізація класу
* Сlass diagram
* Код програми
* Результати детального тестування програми (навести скріншоти виконання тестування програми або скопіювати і вставити у звіт вивід програми на екран)

ХІД РОБОТИ

Опис програми:

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 4), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

1. Програма виконує роль записника завдань, який заповнює користувач в напівавтоматичному режимі.. Програма містить в собі клас Homework, з відповідними для занять характеристиками (будь-який номер завдання(порядковий, чи сам номер завдання), deadline, предмет, тип завдання, текст задання). Більшість з полів є закритими.

private int taskNumber;

private DateTime deadline;

private string subject;

private TaskType taskType;

private string taskText;

public bool done { get; private set; } = false; // автовластивість

1. Для кожного private-поля, яке необхідне для зовнішньої взаємодії, в класі були додані відповідні public-властивості (public- properties). Де необхідна правильність вхідних даних – додано перевірки в секції set властивості. Відповідний програмний код показано нижче.

public int TaskNumber

{

get { return taskNumber; }

set

{

if (value > 0)

taskNumber = value;

else throw new Exception("This value should be above 0");

}

}

DateTime mindeadline = new DateTime(2023, 09, 01);

DateTime maxdeadline = new DateTime(2024, 12, 31);

public DateTime Deadline

{

get { return deadline; }

set

{

if (DateTime.TryParseExact(value.ToString("dd-MM-yyyy"), "dd-MM-yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime parsedDate))

{

if (parsedDate >= mindeadline || parsedDate <= maxdeadline)

deadline = parsedDate;

else throw new Exception($"Date of deadline should be in range {mindeadline} - {maxdeadline}.");

}

else throw new Exception("Invalid date format. Please use the 'dd-mm-yy' for input.");

}

}

public string Subject

{

get { return subject; }

set

{

if (string.IsNullOrEmpty(value) || string.IsNullOrWhiteSpace(value))

{

throw new Exception("Subject can`t be empty or contains numbers.");

}

else

{

if (value.Any(c => char.IsLetter(c) || char.IsSymbol(c)))

subject = value;

}

if (value.Any(c => char.IsLetter(c) || char.IsSymbol(c)))

subject = value;

}

}

public TaskType TaskType

{

get { return taskType; }

set { taskType = value; }

}

public string TaskText

{

get { return taskText; }

set { taskText = value; }

}

public bool IsExpired

{

get { return Deadline < DateTime.Today; }

}

public void IsDone()

{

if (Deadline < DateTime.Today)

{

done = true;

}

}

1. До класу було додано нову автовластивіть – Boolean Done. Відповідає за перевірку чи виконане завдання. Для перевірки використовує порівння за датою. Значення за замовчуванням встановлено **false**.

public bool done { get; private set; } = false; // автовластивість

public void IsDone()

{

if (Deadline < DateTime.Today)

{

done = true;

}

}

1. До класу була додана обчислювальна властивість – IsExpired(термін виконання сплив), яка не пов’язана з будь-яким полем класу і використовує інші властивості та/або методи.

public bool IsExpired

{

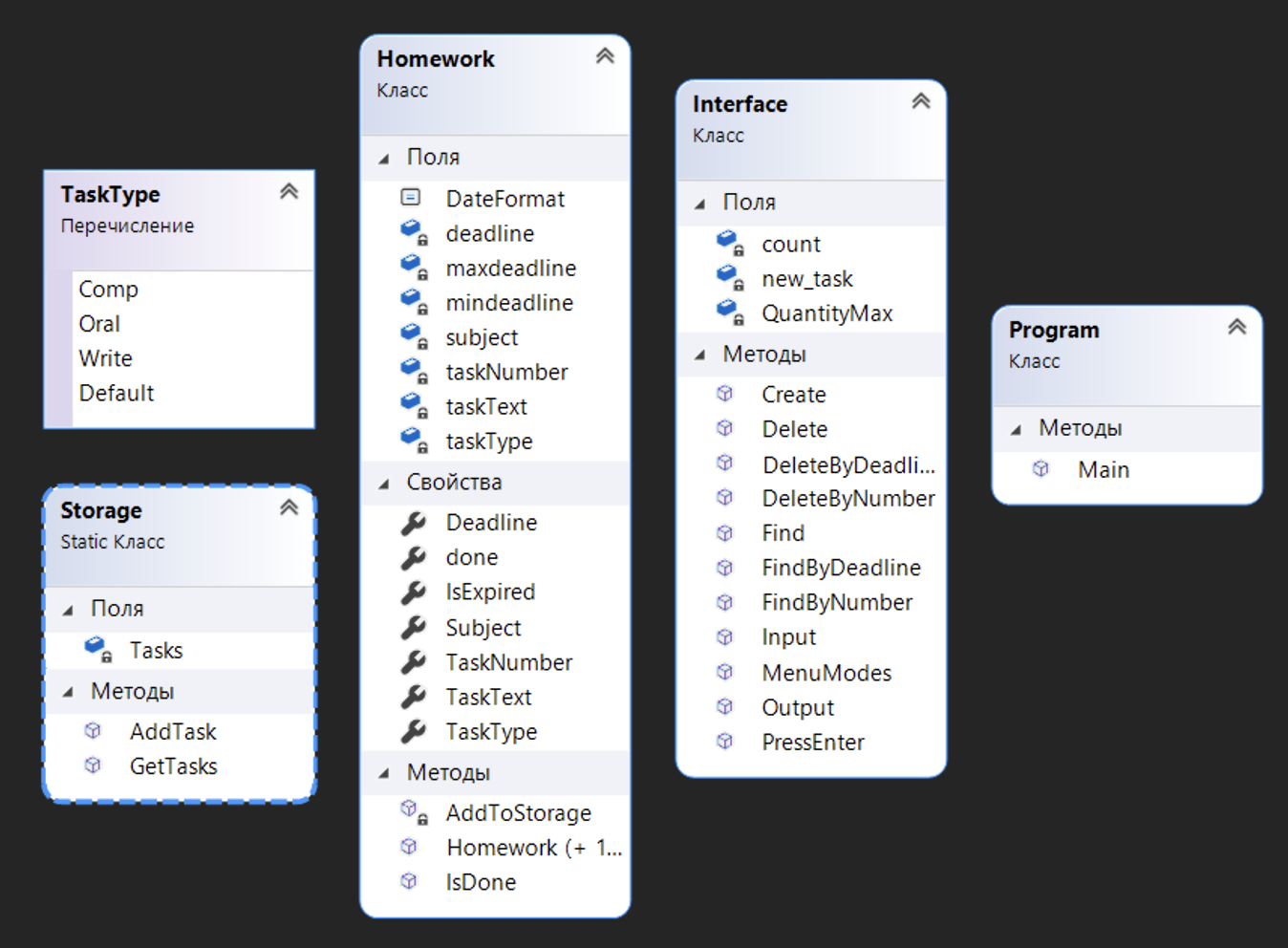
get { return Deadline < DateTime.Today; }

}

1. Властивість done має різній рівень доступу для секцій get і set. Модифікатор доступу секції більш обмежувальний, ніж модифікатор самої властивості.

public bool done { get; private set; } = false; // автовластивість

1. Встановлення та зчитування значень полів в основній програмі реалізувано через додані у клас властивості.
2. Class Diagram

Рисунок 1 – Створена діаграма класу

1. Тестування:

Для початку перевіримо, які значення може прийняти обмежувач кількості завдань (див. рисунок 2)

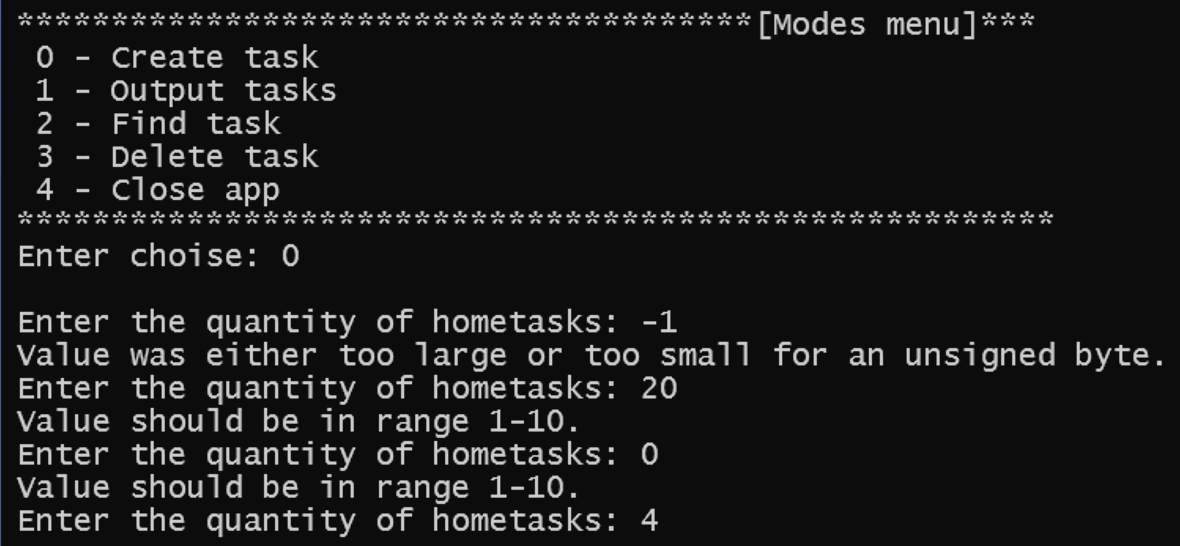


Рисунок 2 – перевірка діапазону введених значень кількості завдань

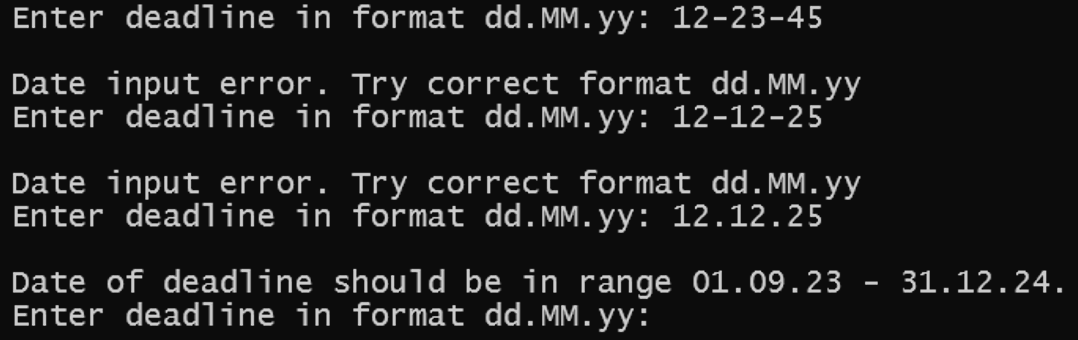


Рисунок 3 – перевірка діапазону введених значень дати

Додамо завдання в список, та перевіримо їх наявність.



Рисунок 4– перевірка введених завдань

Спробуємо додати ще одне завдання, яке не може бути додано через заданий раніше ліміт

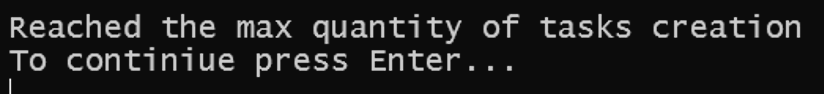


Рисунок 5 – перевірка введення ще одного завдання через ліміт

* 1. Далі спробуємо знайти заняття, які були додані (виконаємо пошук по номеру завдання та дедлайном):

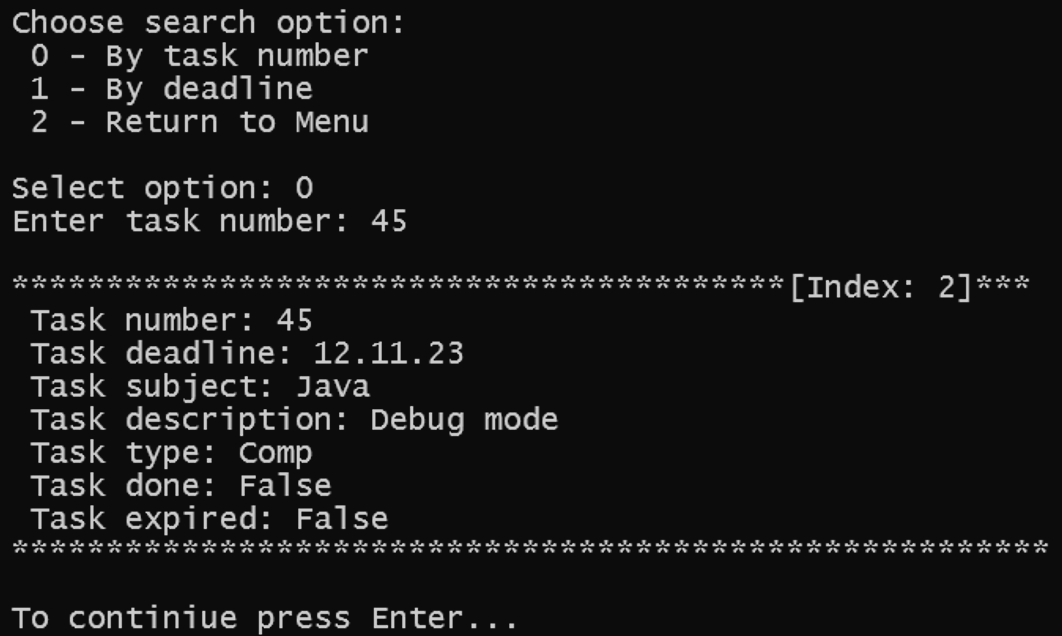
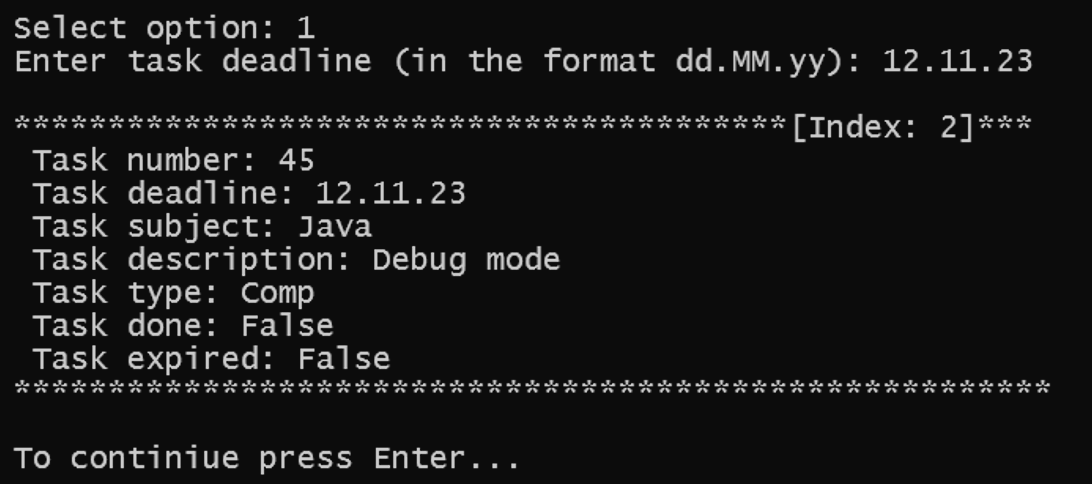


Рисунок 6 – пошук завдання за номером

Рисунок 7 – пошук завдання за дедлайном

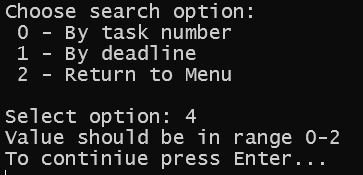


Рисунок 8 – введення некоректного режиму пошуку

Видалимо декілька завдань

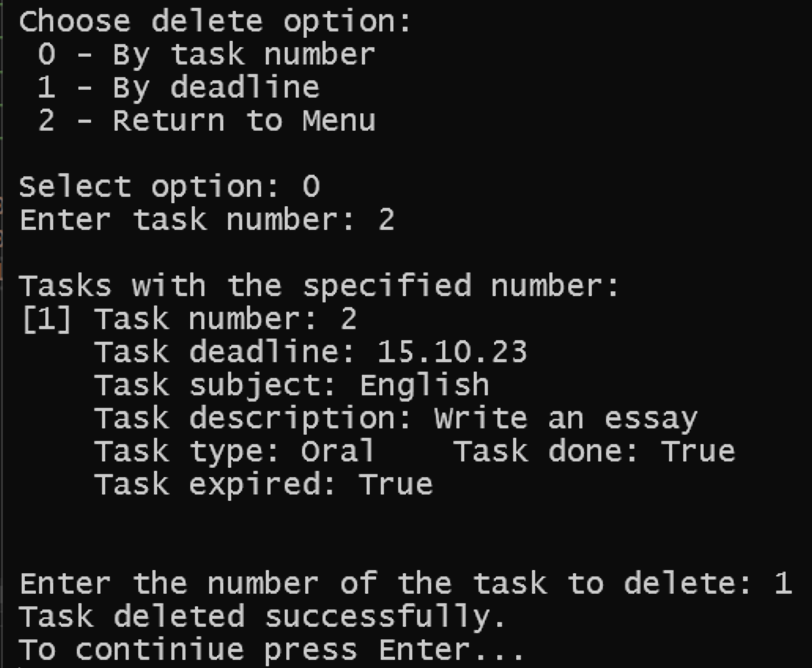


Рисунок 9 – видалення завдання за номером

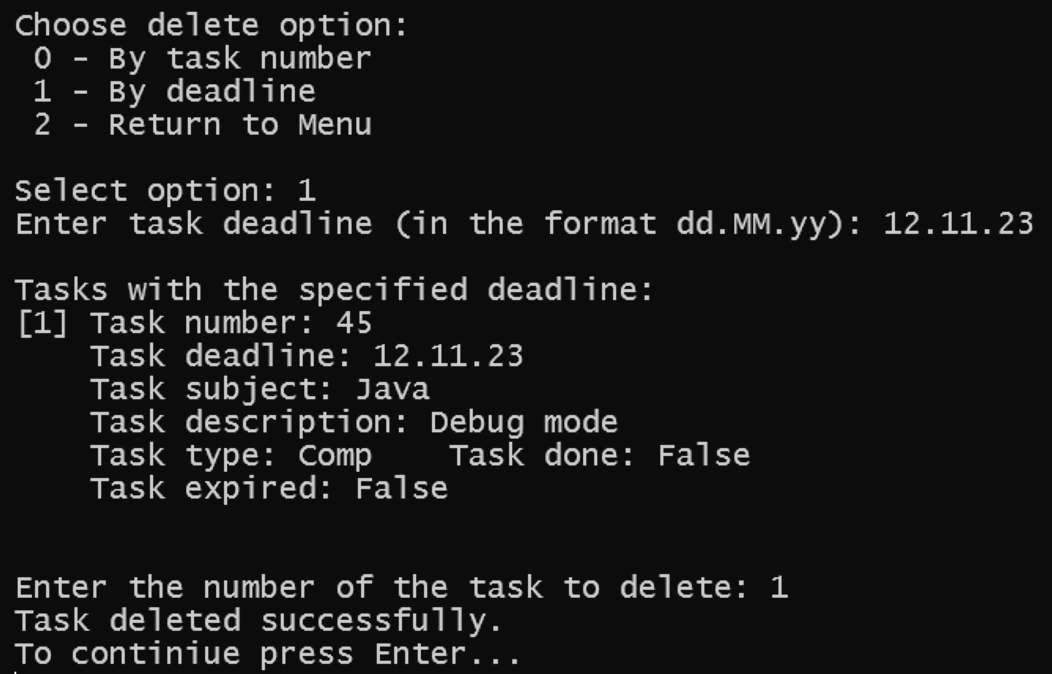
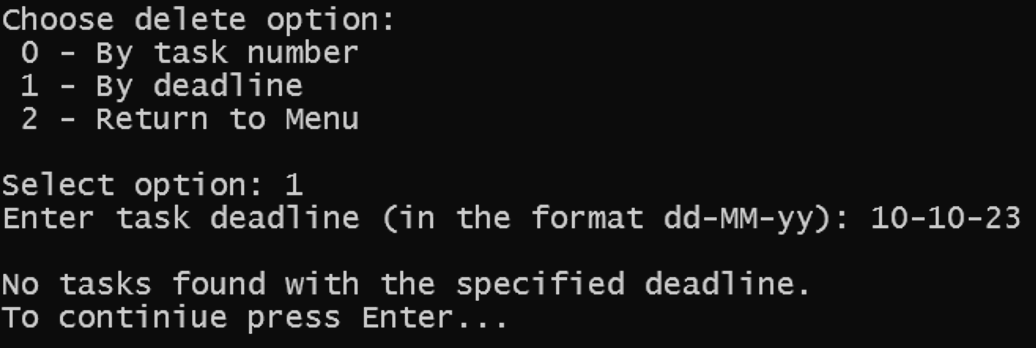
****

Рисунок 10 – видалення завдання за дедлайном

Видалимо всі заняття, які залишились. Також спробуємо видалити заняття, коли їх не існує.

Рисунок 11 – видалення елементів при їх відсутності

# ВИСНОВОК

У ході виконання лабораторної роботи я:

* Навчився інкапсулювати поля класу за допомогою модифікатору доступу private;
* Навчився створювати public властивості для private полів, які необхідні для зовнішньої взаємодії, а також додавання перевірки до секції set;
* До класу було додано автовластивість зі значенням за замовчуванням. Автовластивість спрощує створення властивостей без необхідності явної реалізації get та set методів.
* Була додана обчислювальна властивість, яка не була пов'язана з полем класу. Ця властивість використовувала інші властивості для обчислення значення без прив'язки до конкретного поля.
* Була створена властивість, у якій різний рівень доступу був визначений для секцій get і set. Властивість мала більш обмежений модифікатор доступу в секції set, що дозволило змінювати значення властивості лише в межах класу.
* Встановлення та зчитування значень полів класу було реалізовано через додані у клас властивості. Це дозволило здійснювати зовнішню взаємодію з об'єктом класу за допомогою властивостей, забезпечуючи контроль над доступом до даних.

В результаті виконання завдання було успішно навченося принципам інкапсуляції даних, створенню властивостей та їх різним рівням доступу для секцій get і set, а також роботі з автовластивостями та обчислювальними властивостями в об'єктно-орієнтованому програмуванні.

**ДОДАТОК А***Машинний лістинг програми*

**Program.cs:**

using System.Text;

using System.Threading.Tasks;

namespace Lab\_1

{

public class Program

{

public static void Main(string[] args)

{

// можливість зчитування та виведення кирилічних символів

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

Console.InputEncoding = System.Text.Encoding.GetEncoding(1251);

Console.OutputEncoding = System.Text.Encoding.GetEncoding(1251);

#region debug\_info

//debug list

var homework1 = new Homework(1, DateTime.Parse("2023-10-10"), "Math", TaskType.Write, "Solve equations");

var homework2 = new Homework(2, DateTime.Parse("2023-10-15"), "English", TaskType.Oral, "Write an essay");

#endregion

Interface ui = new();

while (true)

{

ui.MenuModes();

switch (ui.Input("Enter choise: ", 4, 0))

{

case 0: ui.Create(); ui.PressEnter(); break;

case 1: ui.Output(); ui.PressEnter(); break;

case 2: ui.Find(); ui.PressEnter(); break;

case 3: ui.Delete(); ui.PressEnter(); break;

case 4: Environment.Exit(0); ui.PressEnter(); break;

}

}

}

}

}

**ДОДАТОК Б***Машинний лістинг класу*

**Interrface.cs:**

using Microsoft.VisualBasic;

using System.ComponentModel.Design;

using System.Globalization;

using System.Threading.Tasks;

using System.Transactions;

namespace Lab\_1

{

internal class Interface

{

private int QuantityMax;

private int count = Storage.GetTasks().Count;

Homework new\_task = new();

public void Create()

{

TaskType taskType = TaskType.Default;

#region input

while (true)

{

try

{

QuantityMax = Input("Enter the quantity of hometasks: ", 10, 1);

if (count >= QuantityMax) throw new Exception("Reached the max quantity of tasks creation");

break;

}

catch (Exception ex) { Console.WriteLine(ex.Message); break; }

}

//Homework new\_task = new();

while (count < QuantityMax)

{

while (true)

{

try

{

Console.Write("Enter task number: ");

int taskNumber;

int.TryParse(Console.ReadLine(), out taskNumber);

new\_task.TaskNumber = taskNumber;

break;

}

catch (Exception ex) { Console.WriteLine(ex.Message); continue; }

}

while (true)

{

try

{

Console.Write($"Enter deadline in format {Homework.DateFormat}: ");

string input\_date = Console.ReadLine();

Console.WriteLine();

if (DateTime.TryParseExact(input\_date, Homework.DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))

{

new\_task.Deadline = result; break;

}

else throw new Exception($"Date input error. Try correct format {Homework.DateFormat}");

}

catch (Exception ex) { Console.WriteLine(ex.Message); continue; }

}

while (true)

{

try

{

Console.Write("Enter subject: ");

string subject = Console.ReadLine();

new\_task.Subject = subject;

break;

}

catch (Exception ex) { Console.WriteLine(ex.Message); continue; }

}

Console.WriteLine();

while (true)

{

Console.WriteLine($"\*\*\*[Task Types]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n" +

$" 0 - Comp \r\n" +

$" 1 - Oral \r\n" +

$" 2 - Write");

Console.WriteLine($"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

while (true)

{

try

{

//Console.Write("Select task type: ");

//byte select = byte.Parse(Console.ReadLine());

//if (select >= 3) throw new Exception("Value should be in range 0-2");

byte select;

select = Convert.ToByte(Input("Select task type: ", 2, 0));

switch (select)

{

case 0: taskType = TaskType.Comp; break;

case 1: taskType = TaskType.Oral; break;

case 2: taskType = TaskType.Write; break;

default: Console.Write("Input error. Try again"); continue;

}

break;

}

catch (Exception ex) { Console.WriteLine($"{ex.Message}"); }

}

new\_task.TaskType = taskType;

break;

}

while (true)

{

try

{

Console.Write("Enter task description: ");

string taskText = Console.ReadLine();

Console.WriteLine();

new\_task.TaskText = taskText;

break;

}

catch (Exception ex) { Console.WriteLine(ex.Message); continue; }

}

#endregion

Homework task = new(new\_task.TaskNumber, new\_task.Deadline, new\_task.Subject, new\_task.TaskType, new\_task.TaskText);

Console.WriteLine("Task created successfuly\n++++++++++++++++++++++++++");

count++;

}

}

public void Output()

{

foreach (Homework task in Storage.GetTasks())

{

if (!(Storage.GetTasks().Any()))

throw new Exception("No tasks exist in ");

task.IsDone();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index: {Storage.GetTasks().IndexOf(task)}]\*\*\*\n" +

$" Task number: {task.TaskNumber}\n" +

$" Task deadline: {task.Deadline.ToString(Homework.DateFormat)}\n" +

$" Task subject: {task.Subject}\n" +

$" Task description: {task.TaskText}\n" +

$" Task type: {task.TaskType}\n" +

$" Task done: {task.done}\n" +

$" Task expired: {task.IsExpired}\n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

}

public void Find()

{

while (true)

{

Console.WriteLine($"Choose search option: \n" +

$" 0 - By task number\n" +

$" 1 - By deadline\n" +

$" 2 - Return to Menu\n");

Console.Write("Select option: ");

byte select = byte.Parse(Console.ReadLine());

if (select >= 3) Console.WriteLine("Value should be in range 0-2");

switch (select)

{

case 0: FindByNumber(); break;

case 1: FindByDeadline(); break;

case 2: return;

}

break;

}

}

public void FindByNumber()

{

Console.Write("Enter task number: ");

int inputNumber = Convert.ToInt32(Console.ReadLine());

Console.WriteLine();

foreach (Homework task in Storage.GetTasks())

{

if (inputNumber == task.TaskNumber)

{

task.IsDone();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index: {Storage.GetTasks().IndexOf(task)}]\*\*\*\n" +

$" Task number: {task.TaskNumber.ToString()}\n" +

$" Task deadline: {task.Deadline.ToString(Homework.DateFormat)}\n" +

$" Task subject: {task.Subject}\n" +

$" Task description: {task.TaskText}\n" +

$" Task type: {task.TaskType}\n" +

$" Task done: {task.done}\n" +

$" Task expired: {task.IsExpired}\n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

}

}

public void FindByDeadline()

{

bool inputDateValid = false;

while (!inputDateValid)

{

Console.Write($"Enter task deadline (in the format {Homework.DateFormat}): ");

string inputDateString = Console.ReadLine();

DateTime inputDate;

if (DateTime.TryParseExact(inputDateString, Homework.DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out inputDate))

{

inputDateValid = true;

}

else if (!DateTime.TryParseExact(inputDateString, Homework.DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out inputDate))

{

Console.WriteLine($"Invalid date format. Please enter the date in {Homework.DateFormat} format.");

return;

}

Console.WriteLine();

foreach (Homework task in Storage.GetTasks())

{

if (inputDate == task.Deadline)

{

task.IsDone();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index: {Storage.GetTasks().IndexOf(task)}]\*\*\*\n" +

$" Task number: {task.TaskNumber}\n" +

$" Task deadline: {task.Deadline.ToString(Homework.DateFormat)}\n" +

$" Task subject: {task.Subject}\n" +

$" Task description: {task.TaskText}\n" +

$" Task type: {task.TaskType}\n" +

$" Task done: {task.done}\n" +

$" Task expired: {task.IsExpired}\n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

}

}

}

public void Delete()

{

while (true)

{

Console.WriteLine($"Choose delete option: \n" +

$" 0 - By task number\n" +

$" 1 - By deadline\n" +

$" 2 - Return to Menu\n");

Console.Write("Select option: ");

byte select = byte.Parse(Console.ReadLine());

if (select >= 3) throw new Exception("Value should be in range 0-2");

switch (select)

{

case 0: DeleteByNumber(); break;

case 1: DeleteByDeadline(); break;

case 2: return;

}

break;

}

}

public void DeleteByNumber()

{

int inputNumber;

while (true)

{

try

{

Console.Write("Enter task number: ");

inputNumber = Convert.ToUInt16(Console.ReadLine());

if (inputNumber > 0)

{

Console.WriteLine(); break;

}

else throw new Exception("Number shold be above zero");

}

catch (Exception ex) { Console.WriteLine(ex.Message); continue; }

}

List<Homework> tasksToDelete = new List<Homework>();

for (int i = 0; i < Storage.GetTasks().Count; i++)

{

Homework task = Storage.GetTasks()[i];

if (inputNumber == task.TaskNumber)

{

tasksToDelete.Add(task);

}

}

if (tasksToDelete.Count == 0)

{

Console.WriteLine("No tasks found with the specified number.");

return;

}

Console.WriteLine("Tasks with the specified number:");

for (int i = 0; i < tasksToDelete.Count; i++)

{

tasksToDelete[i].IsDone();

Console.WriteLine($"[{i + 1}] Task number: {tasksToDelete[i].TaskNumber}");

Console.WriteLine($" Task deadline: {tasksToDelete[i].Deadline.ToString(Homework.DateFormat)}");

Console.WriteLine($" Task subject: {tasksToDelete[i].Subject}");

Console.WriteLine($" Task description: {tasksToDelete[i].TaskText}");

Console.WriteLine($" Task type: {tasksToDelete[i].TaskType}\n" +

$" Task done: {tasksToDelete[i].done}\n" +

$" Task expired: {tasksToDelete[i].IsExpired}\n");

Console.WriteLine();

}

Console.Write("Enter the number of the task to delete: ");

if (int.TryParse(Console.ReadLine(), out int selectedTaskIndex) && selectedTaskIndex >= 1 && selectedTaskIndex <= tasksToDelete.Count)

{

// Remove the selected task

Storage.GetTasks().Remove(tasksToDelete[selectedTaskIndex - 1]);

Console.WriteLine("Task deleted successfully.");

}

else

{

Console.WriteLine("Invalid input. Task was not deleted.");

}

}

public void DeleteByDeadline()

{

Console.Write($"Enter task deadline (in the format {Homework.DateFormat}): ");

string inputDateString = Console.ReadLine();

DateTime inputDate;

if (!DateTime.TryParseExact(inputDateString, Homework.DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out inputDate))

{

Console.WriteLine($"Invalid date format. Please enter the date in {Homework.DateFormat} format.");

return;

}

Console.WriteLine();

List<Homework> tasksToDelete = new List<Homework>();

// Find tasks with the same deadline and add them to tasksToDelete list

for (int i = 0; i < Storage.GetTasks().Count; i++)

{

Homework task = Storage.GetTasks()[i];

if (inputDate == task.Deadline)

{

tasksToDelete.Add(task);

}

}

if (tasksToDelete.Count == 0)

{

Console.WriteLine("No tasks found with the specified deadline.");

return;

}

Console.WriteLine("Tasks with the specified deadline:");

for (int i = 0; i < tasksToDelete.Count; i++)

{

Console.WriteLine($"[{i + 1}] Task number: {tasksToDelete[i].TaskNumber}");

Console.WriteLine($" Task deadline: {tasksToDelete[i].Deadline.ToString(Homework.DateFormat)}");

Console.WriteLine($" Task subject: {tasksToDelete[i].Subject}");

Console.WriteLine($" Task description: {tasksToDelete[i].TaskText}");

Console.WriteLine($" Task type: {tasksToDelete[i].TaskType}\n" +

$" Task done: {tasksToDelete[i].done}\n" +

$" Task expired: {tasksToDelete[i].IsExpired}\n");

Console.WriteLine();

}

Console.Write("Enter the number of the task to delete: ");

if (int.TryParse(Console.ReadLine(), out int selectedTaskIndex) && selectedTaskIndex >= 1 && selectedTaskIndex <= tasksToDelete.Count)

{

// Remove the selected task

Storage.GetTasks().Remove(tasksToDelete[selectedTaskIndex - 1]);

Console.WriteLine("Task deleted successfully.");

}

else

{

Console.WriteLine("Invalid input. Task was not deleted.");

}

}

#region UI

public void MenuModes()

{

Console.Clear();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Modes menu]\*\*\*\n" +

$" 0 - Create task\n" +

$" 1 - Output tasks\n" +

$" 2 - Find task\n" +

$" 3 - Delete task \n" +

$" 4 - Close app \n" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

public ushort Input(string text, ushort up\_range, ushort down\_range)

{

while (true)

{

ushort input;

try

{

Console.Write(text);

input = Byte.Parse(Console.ReadLine());

if (input > up\_range || input < down\_range)

throw new Exception($"Value should be in range {down\_range}-{up\_range}.");

}

catch (Exception ex)

{

Console.WriteLine($"{ex.Message}");

continue;

}

Console.WriteLine();

return input;

}

}

public void PressEnter()

{

while (true)

{

Console.WriteLine("To continiue press Enter...");

var key = Console.ReadKey(intercept: true);

if (key.Key == ConsoleKey.Enter)

{

Console.Clear();

break; // Выход из цикла, если нажата клавиша Enter

}

else Console.WriteLine("Pressed another key");

}

}

#endregion

}

}

**ДОДАТОК В**

*Машинний лістинг класу*

**Homework.cs:**

using System.Globalization;

using System.Threading.Tasks;

namespace Lab\_1

{

public class Homework

{

public const string DateFormat = "dd.MM.yy";

private int taskNumber;

private DateTime deadline;

private string subject;

private TaskType taskType;

private string taskText;

public bool done { get; private set; } = false; // автовластивість

public Homework() { }

public Homework(int taskNumber, DateTime deadline, string subject, TaskType taskType, string taskText) //конструктор

{

this.taskNumber = taskNumber;

this.deadline = deadline;

this.subject = subject;

this.taskType = taskType;

this.taskText = taskText;

this.AddToStorage();

}

public int TaskNumber

{

get { return taskNumber; }

set

{

if (value > 0)

taskNumber = value;

else throw new Exception("This value should be above 0");

}

}

DateTime mindeadline = new DateTime(2023, 09, 01);

DateTime maxdeadline = new DateTime(2024, 12, 31);

public DateTime Deadline

{

get { return deadline; }

set

{

if (DateTime.TryParseExact(value.ToString(DateFormat), DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime parsedDate))

{

if (parsedDate >= mindeadline && parsedDate <= maxdeadline)

deadline = parsedDate;

else throw new Exception($"Date of deadline should be in range {mindeadline.ToString(DateFormat)} - {maxdeadline.ToString(DateFormat)}.");

}

else throw new Exception($"Invalid date format. Please use the '{DateFormat}' for input.");

}

}

public string Subject

{

get { return subject; }

set

{

if (string.IsNullOrEmpty(value) || string.IsNullOrWhiteSpace(value))

throw new Exception("Subject can`t be empty or contains numbers.");

else if (value.Any(c => char.IsDigit(c)))

throw new Exception("Subject can`t contain numbers.");

else subject = value;

}

}

public TaskType TaskType

{

get { return taskType; }

set { taskType = value; }

}

public string TaskText

{

get { return taskText; }

set { taskText = value; }

}

public bool IsExpired

{

get { return Deadline < DateTime.Today; }

}

public void IsDone()

{

if (Deadline < DateTime.Today)

{

done = true;

}

}

private void AddToStorage()

{

Storage.AddTask(this);

}

}

}

**ДОДАТОК Г***Машинний лістинг класу*

**EnumFile.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_1

{

public enum TaskType

{

Comp,

Oral,

Write,

Default

};

}

ДОДАТОК Ґ  
*Машинний лістинг класу*

**Storage.cs:**

namespace Lab\_1

{

internal static class Storage

{

private static List<Homework> Tasks = new List<Homework>();

public static void AddTask(Homework task)

{

Tasks.Add(task);

}

public static List<Homework> GetTasks()

{

return Tasks;

}

}

}