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 № 3**

з дисципліни « Об’єктно орієнтоване програмування »

*назва дисципліни*

на тему: « Конструктори, перевантаження (overloading)»

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 3 теоретичного матеріалу скорегувати програму для лабораторної роботи № 2 наступним чином:

1. Додати до класу не менше трьох перевантажених конструкторів.

* один з конструкторів має бути без параметрів;
* має бути реалізація хоча б одного конструктора через виклик іншого власного конструктора.

1. Для деякого public-методу класу додати його перевантажені версії.
2. Структура меню залишається з лабораторної роботи № 2:

1 – додати об’єкт

2 – вивести на екран об’єкти

3 – знайти об’єкт

4 – видалити об’єкт

5 – демонстрація поведінки

0 – вийти з програми

При додаванні об’єктів передбачити їх створення різними конструкторами, наприклад, за допомогою виклику випадково обраного конструктора. При обрані конструктора без параметрів передбачити застосування ініціалізаторів.

До програмної реалізації пункту меню «демонстрація поведінки» додати демонстрацію роботи перевантажених методів.

1. 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.
2. Оформити звіт:

* Титульний аркуш
* Завдання
* Сlass diagram
* Програмна реалізація класу
* 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. Class Diagram

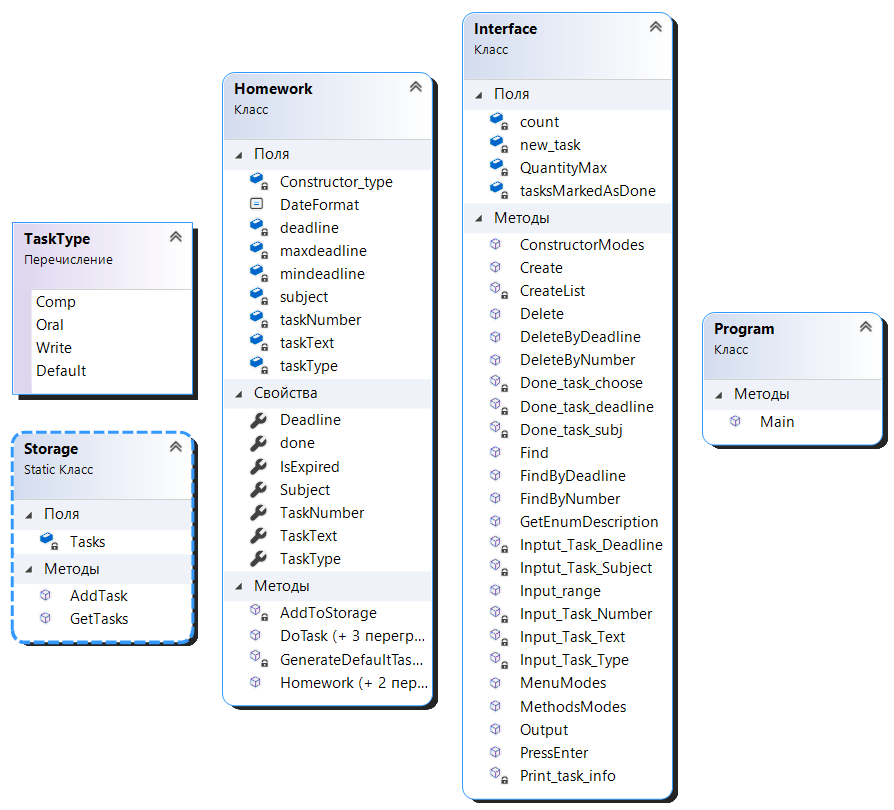


Рисунок 1 – Створена діаграма класу

1. Програмна реалізація класу подана у додатку В.
2. Повний код програми можна переглянути у додатках А, Б, В, Г, Ґ
3. Тестування:

Протестуємо додавання елементів за допомогою різних конструкторів (див. рисунок 2, 3, 4)



Рисунок 2 – створення за допомогою конструктора з введенням всіх полів

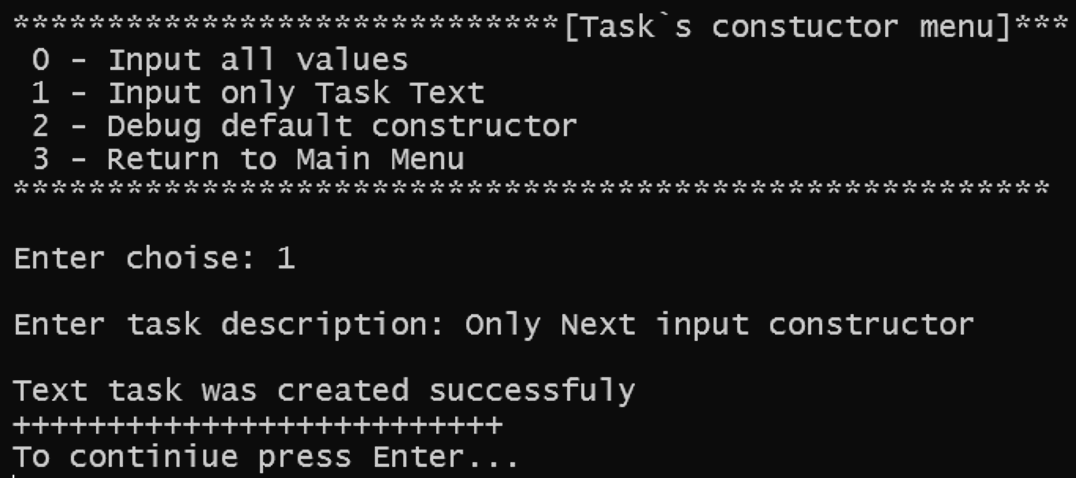


Рисунок 3 – створення за допомогою конструктора з введенням тільки тексту

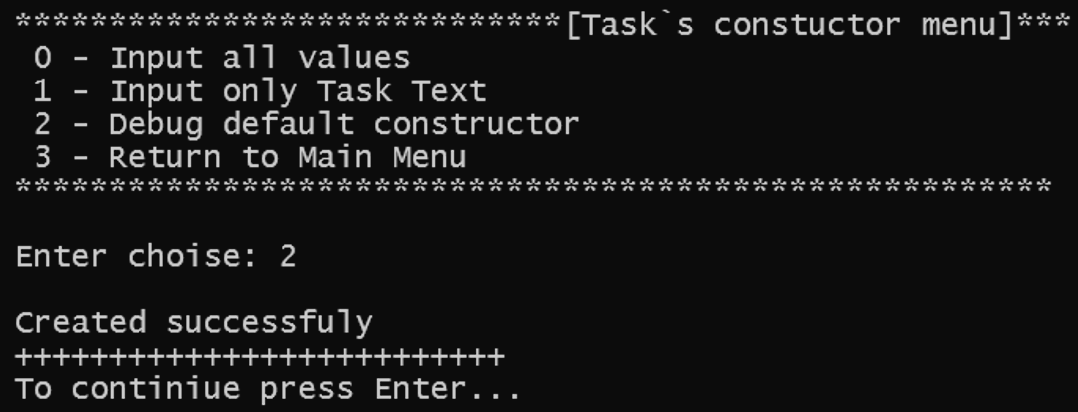


Рисунок 4 – автоматичне створення за допомогою стандартного конструктора

Тепер можемо переглянути весь список створених завдань за допомогою різних конструкторів:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index:0]\*\*\*

Task number: 33

Task deadline: 13.11.23

Task subject: SUBJECT\_DEBUG

Task description: DeBug TeXt)

Task type: Comp

Task done: False

Task expired: False

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index:1]\*\*\*

Task number: 34

Task deadline: 26.10.23

Task subject: ---

Task description: Only Next input constructor

Task type: Default

Task done: False

Task expired: False

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index:2]\*\*\*

Task number: 0

Task deadline: 26.10.23

Task subject: no\_subj

Task description: nothing ¯\\_(ツ)\_/¯

Task type: Default

Task done: False

Task expired: False

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To continiue press Enter...

Після створення списку з завданнями, можемо застосувати до кожного з них створений перевантажений метод **DoTask**. Результат нижче.

Мітка виконання за номером невиконаного завдання по порядку

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Methods menu]\*\*\*

0 - Done task by choose

1 - Done task by Subject

2 - Done task by Deadline

3 - Return to Main Menu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter choise: 0

Undone tasks list:

[1] Task number: 33

Task deadline: 13.11.23

Task subject: SUBJECT\_DEBUG

Task description: DeBug TeXt)

Task type: Comp

Task done: False

Task expired: False

[2] Task number: 34

Task deadline: 26.10.23

Task subject: ---

Task description: Only Next input constructor

Task type: Default

Task done: False

Task expired: False

[3] Task number: 0

Task deadline: 26.10.23

Task subject: no\_subj

Task description: nothing ¯\\_(ツ)\_/¯

Task type: Default

Task done: False

Task expired: False

Enter the number of the task to mark as done: 3

Task marked as done successfully.

To continiue press Enter...

Мітка виконання за предметом невиконаного завдання по порядку

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Methods menu]\*\*\*

0 - Done task by choose

1 - Done task by Subject

2 - Done task by Deadline

3 - Return to Main Menu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter choise: 1

Undone tasks list:

[1] Task number: 33

Task deadline: 13.11.23

Task subject: SUBJECT\_DEBUG

Task description: DeBug TeXt)

Task type: Comp

Task done: False

Task expired: False

[2] Task number: 34

Task deadline: 26.10.23

Task subject: ---

Task description: Only Next input constructor

Task type: Default

Task done: False

Task expired: False

Enter the subject of the task/s to mark as done: SUBJECT\_DEBUG

Marked 1 task(s) as done successfully.

To continiue press Enter...

Мітка виконання за предметом невиконаного завдання по порядку

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Methods menu]\*\*\*

0 - Done task by choose

1 - Done task by Subject

2 - Done task by Deadline

3 - Return to Main Menu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter choise: 2

Undone tasks list:

[1] Task number: 34

Task deadline: 26.10.23

Task subject: ---

Task description: Only Next input constructor

Task type: Default

Task done: False

Task expired: False

Enter the deadline in format dd.MM.yy of the task/s to mark as done: 26.10.23

Marked 1 task(s) as done successfully.

To continiue press Enter...

# ВИСНОВОК

Додав три різних способи створення об'єктів класу. Перший конструктор не потребує параметрів, другий приймає певну кількість параметрів для ініціалізації, і третій використовує виклик іншого конструктора в класі.

Створено кілька версій одного публічного методу з різними параметрами. Для демонстрування роботи програми я створив об'єкти класу за допомогою різних конструкторів, включаючи конструктор без параметрів і використовуючи ініціалізатори. Потім я додав приклади викликів перевантажених методів класу для показу їх роботи. Весь код був ретельно протестований на різних вхідних даних, включаючи некоректні значення, і програма була успішно перевірена на відповідність специфікаціям завдання. На завершення був написаний докладний звіт, який включає опис функцій та результати тестування.

**ДОДАТОК А***Машинний лістинг програми*

**Program.cs:**

using System.Text;

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("2023-10-20"), "English", TaskType.Oral, "Write an essay");

var hm3dbg = new Homework(45, DateTime.Parse("2023-11-21"), "Java", TaskType.Comp, "Debug mode");

var d = new Homework("Debug mode WITH ONLY TASK TEXT");

#endregion

Interface ui = new();

while (true)

{

ui.MenuModes();

switch (ui.Input\_range("Enter choise: ", 5, 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: ui.MethodsModes(); ui.PressEnter(); break;

case 5: Environment.Exit(0); ui.PressEnter(); break;

}

}

}

}

}

**ДОДАТОК Б***Машинний лістинг класу*

**Interrface.cs:**

using System.ComponentModel;

using System.Globalization;

using System.Reflection;

using System.Threading.Tasks;

namespace Lab\_1

{

internal class Interface

{

private int QuantityMax;

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

Homework new\_task = new();

private void Input\_Task\_Number()

{

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; }

}

return;

}

private void Inptut\_Task\_Deadline()

{

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; }

}

}

private void Inptut\_Task\_Subject()

{

while (true)

{

try

{

Console.Write("Enter subject: ");

string subject = Console.ReadLine();

new\_task.Subject = subject;

break;

}

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

}

}

private void Input\_Task\_Type()

{

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\_range("Select task type: ", 2, 0));

switch (select)

{

case 0: new\_task.TaskType = TaskType.Comp; break;

case 1: new\_task.TaskType = TaskType.Oral; break;

case 2: new\_task.TaskType = TaskType.Write; break;

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

}

break;

}

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

}

break;

}

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

}

private void Input\_Task\_Text()

{

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; }

}

}

public void Create()

{

TaskType taskType = TaskType.Default;

while (true)

{

try

{

QuantityMax = Input\_range("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; }

}

ConstructorModes();

while (count < QuantityMax)

{

switch (Input\_range("Enter choise: ", 3, 0))

{

case 0:

{

Input\_Task\_Number();

Inptut\_Task\_Deadline();

Inptut\_Task\_Subject();

Console.WriteLine();

Input\_Task\_Type();

Input\_Task\_Text();

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

Console.WriteLine("Full task was created successfuly\n++++++++++++++++++++++++++");

count++;

break;

}

case 1:

{

Input\_Task\_Text();

Homework task = new(new\_task.TaskText);

Console.WriteLine("Text task was created successfuly\n++++++++++++++++++++++++++");

break;

}

case 2: Homework default\_t = new(); Console.WriteLine("Created successfuly\n++++++++++++++++++++++++++"); Storage.AddTask(default\_t); break;

case 3: MenuModes(); break;

}

break;

}

}

public void Output()

{

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

{

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

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

Print\_task\_info(Storage.GetTasks(), i, 1);

}

}

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();

int i = 0;

foreach (var task in Storage.GetTasks())

{

if (inputNumber == task.TaskNumber)

{

Print\_task\_info(Storage.GetTasks(), i, 1);

}

i++;

}

}

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();

int i = 0;

foreach (var task in Storage.GetTasks())

{

if (inputDate == task.Deadline)

{

Print\_task\_info(Storage.GetTasks(), i, 1);

}

i++;

}

}

}

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>();

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

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++)

{

Print\_task\_info(tasksToDelete, i, 2);

}

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++)

{

Print\_task\_info(tasksToDelete, i, 2);

}

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 - Methods menu \n" +

$" 5 - Close app \n" +

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

}

public void MethodsModes()

{

Console.Clear();

Console.WriteLine("" +

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

$" 0 - Done task by choose\n" +

$" 1 - Done task by Subject\n" +

$" 2 - Done task by Deadline\n" +

$" 3 - Return to Main Menu\n" +

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

switch (Input\_range("Enter choise: ", 3, 0))

{

case 0: Done\_task\_choose(); break;

case 1: Done\_task\_subj(); break;

case 2: Done\_task\_deadline(); break;

case 3: break;

}

}

public void ConstructorModes()

{

Console.Clear();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Task`s constuctor menu]\*\*\*\n" +

$" 0 - Input all values\n" +

$" 1 - Input only Task Text\n" +

$" 2 - Debug default constructor\n" +

$" 3 - Return to Main Menu\n" +

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

}

#region overloaded\_methods

private void Done\_task\_choose()

{

//List<Homework> undone\_tsks = Create\_undone\_lst();

List<Homework> undone\_tsks = CreateList(task => task.done == false);

Console.WriteLine("Undone tasks list:");

for (int i = 0; i < undone\_tsks.Count; i++)

{

Print\_task\_info(undone\_tsks, i, 2);

}

Console.Write("Enter the number of the task to mark as done: ");

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

{

Homework selected\_task = Storage.GetTasks()[selectedTaskIndex - 1];

selected\_task.DoTask(selected\_task.TaskNumber);// Done the selected task

Console.WriteLine("Task marked as done successfully.");

}

else

{

Console.WriteLine("Invalid input. Task was not marked as done. Try again");

}

}

int tasksMarkedAsDone = 0;

private void Done\_task\_subj()

{

//List<Homework> undone\_tsks = Create\_undone\_lst();

List<Homework> undone\_tsks = CreateList(task => task.done == false);

Console.WriteLine("Undone tasks list:");

for (int i = 0; i < undone\_tsks.Count; i++)

{

Print\_task\_info(undone\_tsks, i, 2);

}

Console.Write("Enter the subject of the task/s to mark as done: ");

string input\_subj = Console.ReadLine();

if (!int.TryParse(input\_subj, out int selected\_subj))

{

// Введений рядок не є числом (це можуть бути символи або комбінація символів і цифр)

tasksMarkedAsDone = 0;

foreach (var task in undone\_tsks)

{

if (task.DoTask(input\_subj)) // Done the selected subject task

tasksMarkedAsDone++;

}

if (tasksMarkedAsDone > 0) Console.WriteLine($"Marked {tasksMarkedAsDone} task(s) as done successfully.");

else

{

Console.WriteLine("No matching tasks found for the entered deadline.");

}

}

else

{

Console.WriteLine("Invalid input. Any task was not marked as done. Try again");

}

}

private void Done\_task\_deadline()

{

List<Homework> undone\_tsks = CreateList(task => task.done == false);

Console.WriteLine("Undone tasks list:");

for (int i = 0; i < undone\_tsks.Count; i++)

{

Print\_task\_info(undone\_tsks, i, 2);

}

Console.Write($"Enter the deadline in format {Homework.DateFormat} of the task/s to mark as done: ");

string input\_date = Console.ReadLine();

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

{

// Введена дата в правильному форматі

tasksMarkedAsDone = 0;

foreach (var task in undone\_tsks)

{

if (task.DoTask(inputDate))

tasksMarkedAsDone++;

}

if (tasksMarkedAsDone > 0)

{

Console.WriteLine($"Marked {tasksMarkedAsDone} task(s) as done successfully.");

}

else

{

Console.WriteLine("No matching tasks found for the entered deadline.");

}

}

else

{

Console.WriteLine("Invalid date format. No tasks were marked as done. Please use the correct date format.");

}

}

private List<Homework> CreateList(Func<Homework, bool> condition)

{

List<Homework> filtered\_tasks = new List<Homework>();

foreach (Homework task in Storage.GetTasks())

{

if (condition(task))

{

filtered\_tasks.Add(task);

}

}

return filtered\_tasks;

}

#endregion

private void Print\_task\_info(List<Homework> t, int id, int var)

{

t[id].DoTask();

if (var == 1)

Console.Write($"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Index:{id}]\*\*\*\n ");

else if (var == 2)

Console.Write($"[{id + 1}] ");

Console.WriteLine($"Task number: {t[id].TaskNumber}\n" +

$" Task deadline: {t[id].Deadline.ToString(Homework.DateFormat)}\n" +

$" Task subject: {t[id].Subject} \n" +

$" Task description: {t[id].TaskText}\n" +

$" Task type: {t[id].TaskType}\n" +

$" Task done: {t[id].done}\n" +

$" Task expired: {t[id].IsExpired}");

if (var == 1) Console.WriteLine($"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

else if (var == 2)

Console.WriteLine();

}

public ushort Input\_range(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;

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

#region Constructors\_Lab3

public Homework() // без параметрів

{

this.taskNumber = 0;

this.deadline = DateTime.Today;

this.subject = "no\_subj";

this.taskType = TaskType.Default;

this.taskText = "nothing ¯\\\_(ツ)\_/¯";

this.DoTask();

//this.AddToStorage();

}

//конструктор, який викликає інший конструктор класу

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

: this()

{

this.taskNumber = taskNumber;

this.deadline = deadline;

this.subject = subject;

this.taskType = taskType;

this.taskText = taskText;

this.DoTask();

this.AddToStorage();

}

//конструктор тільки з текстом завдання

public Homework(string taskText)

: this(GenerateDefaultTaskNumber(), DateTime.Today, "---", TaskType.Default, taskText)

{

this.done = false;

this.taskText = taskText;

}

private static int GenerateDefaultTaskNumber()

{

int max\_num = -1;

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

{

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

if (max\_num < tsk.TaskNumber)

max\_num = tsk.TaskNumber;

}

return max\_num + 1;

}

#endregion

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.");

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; }

}

#region overloaded\_methods

public void DoTask()// Реалізація методу без параметрів

{

if (Deadline < DateTime.Today)

{

done = true;

}

}

public void DoTask(int nuber\_of\_tsk)// Реалізація методу з параметром номеру завдання

{

if (this.TaskNumber == nuber\_of\_tsk)

this.done = true;

}

public bool DoTask(string subj)// Реалізація методу з параметром предмету

{

if (this.Subject == subj)

{

this.done = true;

return true;

}

else return false;

}

public bool DoTask(DateTime date)// Реалізація методу з параметром терміну виконання

{

if (this.Deadline == date)

{

this.done = true;

return true;

}

else return false;

}

#endregion

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;

}

}

}