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 № 5**

з дисципліни « Об’єктно орієнтоване програмування »

*назва дисципліни*

на тему: «Модульне тестування (unit testing)»

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 5 теоретичного матеріалу виконати модульне тестування (unit testing) розробленого класу, який був вдосконалений у лабораторній роботі № 4.

1. У Solution, в якому реалізовано основний проєкт (реалізація лабораторної роботи No 4), додати проєкт для побудови модульних тестів MSTest.
2. Для тест-класу MSTest проєкта змінити назву за замовчуванням на відповідну назву.
3. Розробити тест-методи для повного тестування логіки роботи всіх нестатичних і статичних методів класу. назва тест-методів має відображати суть тестів. Структура тестів має відповідати шаблону ААА (Arrange, Act, Assert). В тест-методах має бути продемонстровано використання різних методів класу Assert: AreEqual, IsTrue/IsFalse, ThrowsException та ін.
4. Запустити виконання розроблених тестів. Якщо деякі тести будуть непройдені (fail), то необхідно підкорегувати код методу, який тестується, і досягти повного проходження всіх розроблених тестів.
5. Оформити звіт:

* Титульний аркуш
* Завдання
* Сlass diagram (для основного проєкту і тест-проєкту)
* Реалізація класу
* Реалізація тест-класу
* Код програми
* Результати запуску розроблених тестів

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).

Class Diagram основної програми

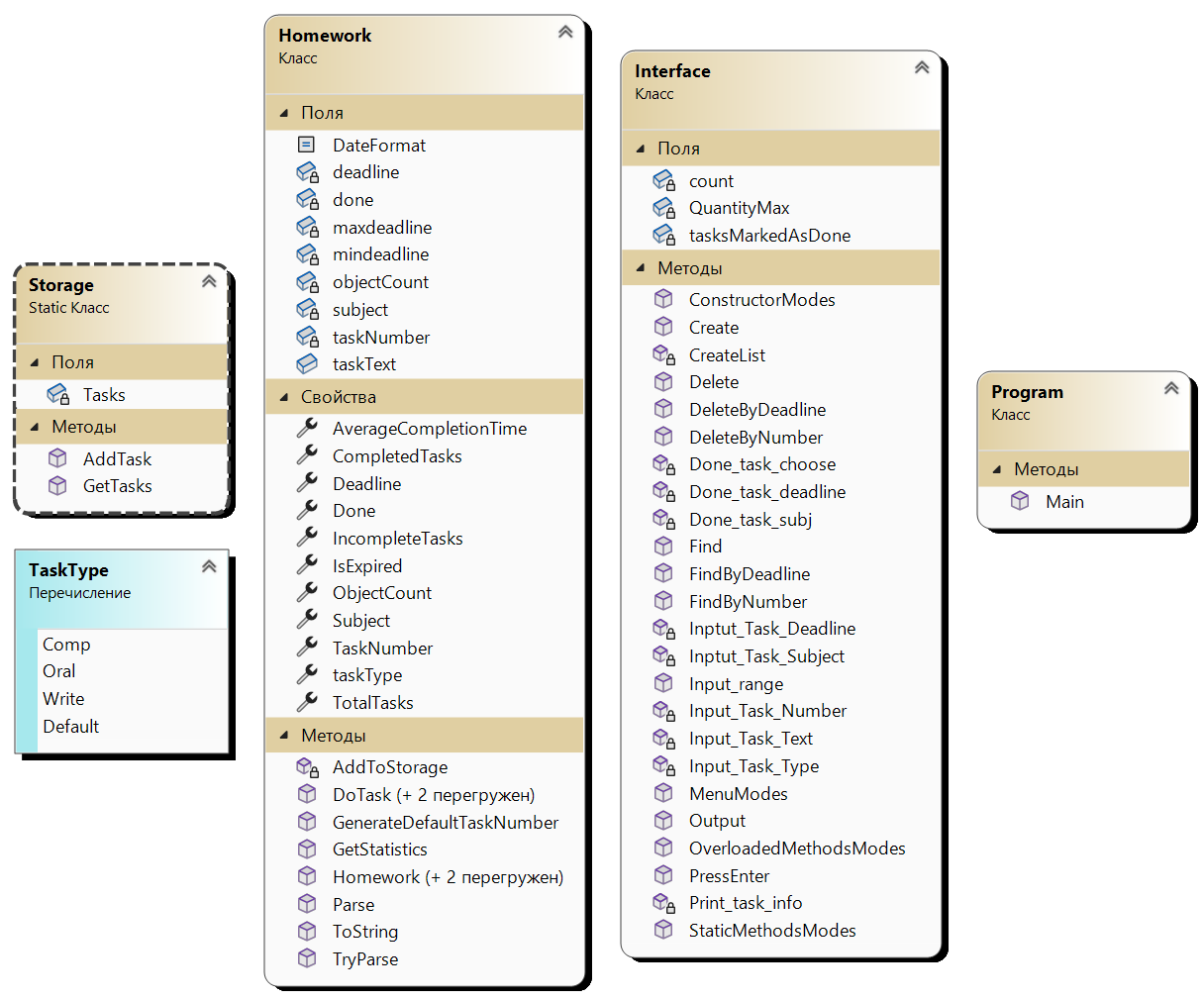


Рисунок 1 – Створена детальна діаграма класів

Class Diagram класу для Unit-тестування

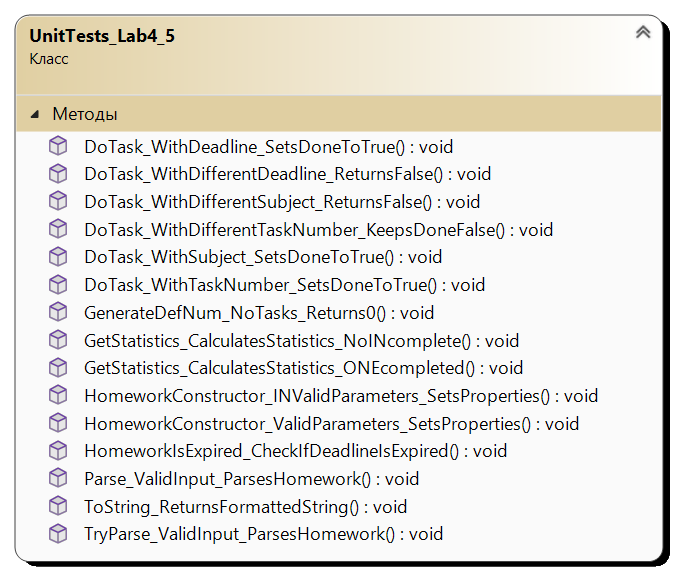


Рисунок 2 – діаграма класів тестування

Реалізація класу

Детальний код класу програми можна переглянути в додатку В

Реалізація тест-класу

Детальний код тест-класу можна переглянути в додатку Д

Код програми

Повний код програми та класів можна переглянути в додатках А, Б, В, Г, Ґ, Д

Результати Unit-тестування:

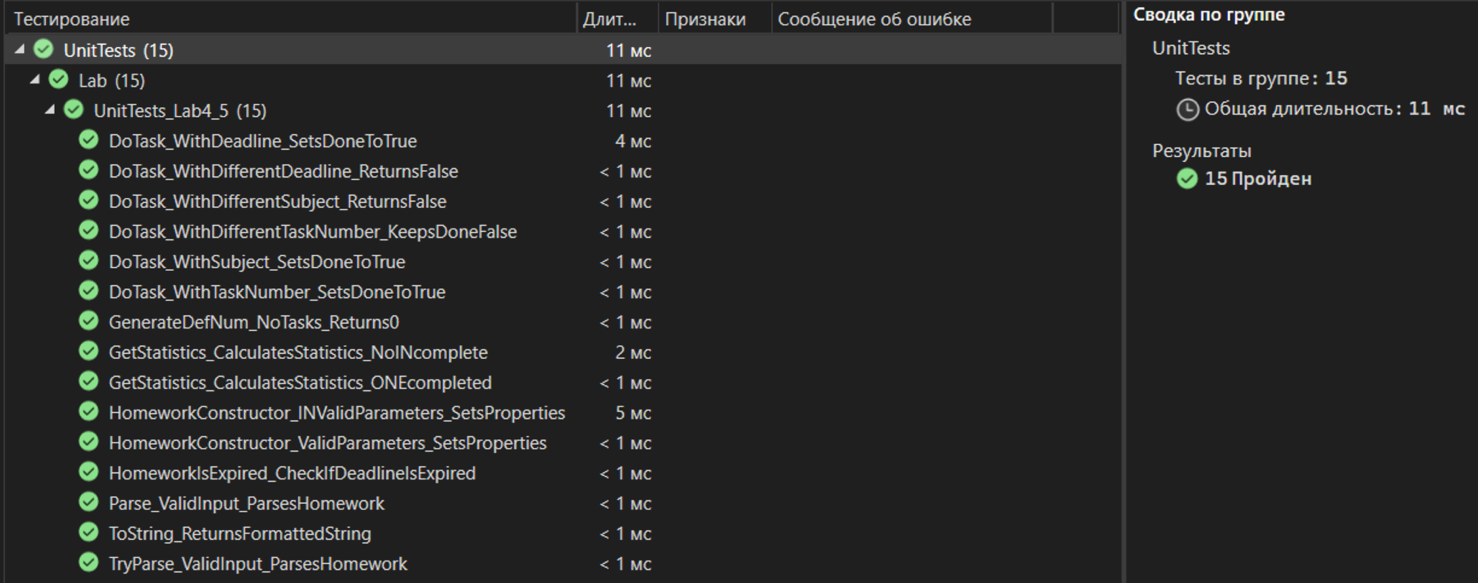


Рисунок 3 – результат виконання написаних unit-тестів для створеного класу

# ВИСНОВОК

У процесі виконання модульного тестування розробленого класу, який був вдосконалений у лабораторній роботі № 4, було виконано наступні кроки.

Додано проєкт для модульних тестів MSTest в Solution основного проєкту. Назва тест-класу MSTest була відповідно змінена для відображення призначення тестів.

Були розроблені тест-методи для всіх нестатичних і статичних методів класу. Кожен тест-метод був структурований відповідно до шаблону ААА (Arrange, Act, Assert). Використано різні методи класу Assert, такі як AreEqual, IsTrue/IsFalse, ThrowsException, для впевнення в коректності роботи кожного методу.

Запущено виконання розроблених тестів. В разі непройдених тестів вносились відповідні корективи в код методів для досягнення повного проходження всіх тестів.

Програма була успішно протестована, і всі розроблені тест-методи успішно виконались. Всі тести вказали на коректність реалізації функціональності класу та його методів.

Модульне тестування сприяло виявленню та усуненню потенційних помилок, підвищило надійність програмного продукту та забезпечило відповідність функціоналу визначеним специфікаціям. З впровадженням модульних тестів покращено якість та стабільність розробленого класу.

***Додаток А***Машинний лістинг програми **Program.cs**

using System.Text;

using System.Text.Unicode;

namespace Lab

{

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);

Console.OutputEncoding = System.Text.Encoding.UTF8;

//

#region debug\_info

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 hm3db = new Homework(45, DateTime.Parse("2023-11-22"), "Java", TaskType.Comp, "Debug mode2");

var hm3 = new Homework(45, DateTime.Parse("2023-11-23"), "Java", TaskType.Comp, "Debug mode3");

var d2 = new Homework("Debug mode WITH ONLY TASK TEXT");

//debug separator "|" input 777 | 11.11.23 | Space Subj | Oral | Debug separator

//

#endregion

Interface ui = new();

while (true)

{

ui.MenuModes();

switch (ui.Input\_range("Enter choise: ", 6, 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.OverloadedMethodsModes(); ui.PressEnter(); break;

case 5: ui.StaticMethodsModes(); ui.PressEnter(); break;

case 6: Environment.Exit(0); ui.PressEnter(); break;

}

}

}

}

}

***Додаток Б***Машинний лістинг класу **Interrface.cs:**

using System.Globalization;

namespace Lab

{

internal class Interface

{

private int QuantityMax;

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

private int Input\_Task\_Number()

{

while (true)

{

try

{

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

int taskNumber;

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

return taskNumber;

}

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

}

}

private DateTime 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))

{

return result;

}

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

}

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

}

}

private string Inptut\_Task\_Subject()

{

while (true)

{

try

{

Console.Write("Enter subject: ");

string subject = Console.ReadLine();

return subject;

}

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

}

}

private TaskType 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

{

byte select;

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

switch (select)

{

case 0: return TaskType.Comp; break;

case 1: return TaskType.Oral; break;

case 2: return TaskType.Write; break;

default: throw new Exception("Input error. Try again"); continue;

}

}

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

}

}

}

private string Input\_Task\_Text()

{

while (true)

{

try

{

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

string taskText = Console.ReadLine();

Console.WriteLine();

return taskText;

}

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

}

}

public void Create()

{

try

{

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; }

}

while (count < QuantityMax)

{

ConstructorModes();

bool back = false;

switch (Input\_range("Enter choise: ", 4, 0))

{

case 0:

{

int n = Input\_Task\_Number();

DateTime inp = Inptut\_Task\_Deadline();

string s = Inptut\_Task\_Subject();

Console.WriteLine();

TaskType t = Input\_Task\_Type();

string txt = Input\_Task\_Text();

Homework task = new(n, inp, s, t, txt);

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

PressEnter();

count++;

break;

}

case 1:

{

string txt = Input\_Task\_Text();

Homework task = new(txt);

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

PressEnter();

count++;

break;

}

case 2:

Homework default\_t = new(); Console.WriteLine("Default created successfuly\n++++++++++++++++++++++++++"); PressEnter();

count++; break;

case 3:

{

bool ok = false;

do

{

try

{

Console.WriteLine($"Input string with:\nTask Number,Deadline (in format {Homework.DateFormat}),Subject,Task Type(Comp,Oral,Write),TaskText\nseparated by \"|\" symbol");

string input = Console.ReadLine();

Exception error;

if (Homework.TryParse(input, out Homework homeworkObject, out error))

{

//немає необхідності тут створювати об'єкт, оскілки в завданні треба щоб трай парс та трай створювали об'єкт класу

//new Homework(homeworkObject.TaskNumber, homeworkObject.Deadline, homeworkObject.Subject, homeworkObject.taskType, homeworkObject.TaskText);

Console.WriteLine("String input by \"|\" separator was created successfuly\n++++++++++++++++++++++++++");

count++; ok= true;

break;

}

else if (error != null)

{

ok = false;

Console.WriteLine("Failed to parse the input. Please check the format and try again.");

throw error; break;

}

}

catch (Exception ex) { Console.WriteLine(ex.Message.ToString()+"\n"); }

}while (!ok);

break;

}

case 4: MenuModes(); back = true; break;

}

if (back == true) break;

}

}

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

}

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");

switch (Input\_range("Select option: ",2,0))

{

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.");

Homework.ObjectCount--;

}

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.");

Homework.ObjectCount--;

}

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 - Overloaded Methods menu \n" +

$" 5 - Static Methods menu \n" +

$" 6 - Close app \n" +

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

}

public void OverloadedMethodsModes()

{

Console.Clear();

Console.WriteLine("" +

$"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*[Overloaded 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 - Create by \"|\" separated string in order\n\"Task Number,Deadline (in format{Homework.DateFormat}),Subject,Task Type(Comp,Oral,Write),TaskText\"\n" +

$" -----------------------------\n" +

$" 4 - Return to Main Menu\n" +

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

}

public void StaticMethodsModes()

{

Console.Clear();

Console.WriteLine("" +

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

$" 0 - Get quantity of all homeworks\n" +

$" 1 - Get statistics\n" +

$" 2 - Return to Main Menu\n" +

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

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

{

case 0: int q = Homework.ObjectCount; Console.WriteLine($"Quantity of hometask is {q}"); break;

case 1:

{

List<Homework> tasks = Storage.GetTasks(); // Отримуємо список завдань з Storage

Homework.GetStatistics(tasks); // Оновлюємо статистику для завдань

Console.WriteLine("Statistics:"); // Виводимо статистику

Console.WriteLine($"Total amount of tasks: {Homework.TotalTasks}");

Console.WriteLine($"Amount of completed tasks: {Homework.CompletedTasks}");

Console.WriteLine($"Amount of INcompleted tasks: {Homework.IncompleteTasks}");

TimeSpan averageTime = Homework.AverageCompletionTime; // Отримуємо середній термін виконання

averageTime = TimeSpan.FromTicks(Math.Abs(averageTime.Ticks)); // модуль для завжди позитивного значення

int days = averageTime.Days; // Отримуємо кількість днів та годин

int hours = averageTime.Hours;

Console.WriteLine($"The average completion time: {days} days and {hours} hours"); break;

}

case 2: break;

}

}

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

{

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

#region overloaded\_methods

int tasksMarkedAsDone = 0;

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");

}

}

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 = 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 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

}

}

***Додаток В***

Машинний лістинг класу **Homework.cs:**

using System;

using System.Globalization;

using System.Threading.Tasks;

namespace Lab

{

public class Homework

{

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

private int taskNumber;

private DateTime deadline;

private string subject;

private bool done;

public string taskText;

public TaskType taskType { get; set; } = TaskType.Default; //автовластивість

// Статичне поле для лічильника об'єктів

private static int objectCount = 0;

// поля для моєї властивості - статистики по виконанню

public static int TotalTasks { get; private set; } = Storage.GetTasks().Count();

public static int CompletedTasks { get; private set; } = 0;

public static int IncompleteTasks => TotalTasks - CompletedTasks;

public static TimeSpan AverageCompletionTime { get; private set; } = TimeSpan.Zero;

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

{

this.taskNumber = taskNumber;

this.deadline = deadline;

this.subject = subject;

this.taskType = taskType;

this.taskText = taskText;

this.AddToStorage();

objectCount++;

}

//конструктор тільки з текстом завдання

public Homework(string taskText)//конструктор, який викликає інший конструктор класу

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

{

this.done = false;

}

// без параметрів

public Homework() //конструктор, який викликає інший конструктор класу

: this(GenerateDefaultTaskNumber(), DateTime.Today, "no subject", TaskType.Default, "nothing ")

{ }

public 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;

}

// Статична властивість для отримання кількості створених об'єктів

public static int ObjectCount

{

get { return objectCount; }

set

{

if (value >= 0)

objectCount = value;

else

objectCount = 0;

}

}

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 bool IsExpired

{

get { return Deadline < DateTime.Today; }

}

public bool Done

{

get { return done; }

private set

{

if (Deadline < DateTime.Today)

done = true;

}

}

#region overloaded\_methods

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);

}

public static Homework Parse(string s)

{

s = s.Trim();

string[] parts = s.Split('|'); // Розділити рядок за роздільником

if (parts.Length != 5)

{

throw new FormatException("Invalid input format. Expected 5 \"|\"-separated values.");

}

// Розбір параметрів та створення об'єкта Homework

int taskNumber;

if (!int.TryParse(parts[0], out taskNumber))

{

throw new FormatException("Invalid Task Number format.");

}

DateTime deadline;

//string tempdate = parts[1].Trim();

if (!DateTime.TryParseExact(parts[1].Trim(), DateFormat, CultureInfo.InvariantCulture, DateTimeStyles.None, out deadline))

{

throw new FormatException($"Invalid deadline format. Please use the '{DateFormat}' format.");

}

string subject = parts[2].Trim(); // Видалення зайвих пробілів

TaskType taskType;

if (!Enum.TryParse(parts[3], out taskType))

{

throw new FormatException("Invalid Task Type format.");

}

string taskText = parts[4].Trim(); // Видалення зайвих пробілів

// Створення та ініціалізація об'єкта Homework

Homework hw = new Homework(taskNumber, deadline, subject, taskType, taskText);

return hw;

}

public static bool TryParse(string s, out Homework obj, out Exception errorMessage)

{

try

{

obj = Parse(s);

errorMessage = null; // Успешный разбор, ошибка отсутствует

return true;

}

catch (Exception ex)

{

obj = null;

errorMessage = ex;

return false;

}

}

public override string ToString()

{

return $"Task Number: {TaskNumber} | Deadline: {Deadline.ToString(DateFormat)} | Subject: {Subject} | Task Type: {taskType} | Task Text: {taskText}";

}

//static method

public static void GetStatistics(List<Homework> tasks)

{

TotalTasks = Storage.GetTasks().Count();

CompletedTasks = tasks.Count(task => task.done);

if (CompletedTasks > 0)

{

AverageCompletionTime = new TimeSpan(tasks.Where(task => task.done)

.Select(task => (DateTime.Today-task.Deadline))

.Sum(timeSpan => timeSpan.Ticks) / CompletedTasks);

}

}

}

}

***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

{

public enum TaskType

{

Comp,

Oral,

Write,

Default

};

}

ДОДАТОК Ґ  
*Машинний лістинг класу* Storage.cs:

namespace Lab

{

public 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;

}

}

}

***Додаток Д***Машинний лістинг програми **UnitTestLab4-5.cs**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Globalization;

namespace Lab

{

[TestClass]

public class UnitTests\_Lab4\_5

{

[TestMethod]

public void HomeworkConstructor\_ValidParameters\_SetsProperties()

{

// Arrange

int taskNumber = 1;

DateTime deadline = DateTime.Today;

string subject = "Math";

TaskType taskType = TaskType.Default;

string taskText = "Complete the math assignment";

// Act

Homework homework = new Homework(taskNumber, deadline, subject, taskType, taskText);

// Assert

Assert.AreEqual(taskNumber, homework.TaskNumber);

Assert.AreEqual(deadline, homework.Deadline);

Assert.AreEqual(subject, homework.Subject);

Assert.AreEqual(taskType, homework.taskType);

Assert.AreEqual(taskText, homework.taskText);

}

[TestMethod]

public void HomeworkConstructor\_INValidParameters\_SetsProperties()

{

// Arrange

int taskNumber = -5;

DateTime deadline = DateTime.ParseExact("11.08.23", Homework.DateFormat, CultureInfo.InvariantCulture);

string subjectn = "";

TaskType taskType = TaskType.Default;

string subject = "20th century";

Homework homeworkn = new Homework(taskNumber, deadline, subjectn, taskType, "empty txt");

Homework homework = new Homework(taskNumber, deadline, subject, taskType, "empty txt");

// Act + Assert

Assert.ThrowsException<Exception>(() => homeworkn.TaskNumber = taskNumber);

Assert.ThrowsException<Exception>(() => homeworkn.Deadline = deadline);

Assert.ThrowsException<Exception>(() => homeworkn.Subject = subjectn);

Assert.ThrowsException<Exception>(() => homework.TaskNumber = taskNumber);

Assert.ThrowsException<Exception>(() => homework.Deadline = deadline);

Assert.ThrowsException<Exception>(() => homework.Subject = subject);

}

[TestMethod]

public void GenerateDefNum\_NoTasks\_Returns0()

{ // Arrange

Storage.GetTasks().Clear();

int expected = 0;

// Act

int d = Storage.GetTasks().Count;

int actual = Homework.GenerateDefaultTaskNumber();

// Assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void HomeworkIsExpired\_CheckIfDeadlineIsExpired()

{

// Arrange

DateTime pastDeadline = DateTime.Today.AddDays(-1);

DateTime today = DateTime.Today;

DateTime futureDeadline = DateTime.Today.AddDays(1);

Homework homework = new Homework();

// Act

homework.Deadline = pastDeadline;

// Assert

Assert.IsTrue(homework.IsExpired, "Deadline is in the past, so IsExpired should be true.");

// Act

homework.Deadline = today;

// Assert

Assert.IsFalse(homework.IsExpired, "Deadline is in the past, so IsExpired should be false.");

// Act

homework.Deadline = futureDeadline;

// Assert

Assert.IsFalse(homework.IsExpired, "Deadline is in the future, so IsExpired should be false.");

}

[TestMethod]

public void DoTask\_WithTaskNumber\_SetsDoneToTrue()

{

// Arrange

Homework homework = new Homework(1, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment");

// Act

homework.DoTask(1);

// Assert

Assert.IsTrue(homework.Done);

}

[TestMethod]

public void DoTask\_WithDifferentTaskNumber\_KeepsDoneFalse()

{

// Arrange

Homework homework = new Homework(1, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment");

// Act

homework.DoTask(2);

// Assert

Assert.IsFalse(homework.Done);

}

[TestMethod]

public void DoTask\_WithSubject\_SetsDoneToTrue()

{

// Arrange

Homework homework = new Homework(1, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment");

// Act

bool actual = homework.DoTask("Math");

// Assert

Assert.AreEqual(actual, true);

Assert.IsTrue(homework.Done);

}

[TestMethod]

public void DoTask\_WithDifferentSubject\_ReturnsFalse()

{

// Arrange

Homework homework = new Homework(1, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment");

// Act

bool actual = homework.DoTask("English");

// Assert

Assert.IsFalse(actual);

Assert.IsFalse(homework.Done);

}

[TestMethod]

public void DoTask\_WithDeadline\_SetsDoneToTrue()

{

// Arrange

DateTime tommorow = DateTime.Today.AddDays(1);

Homework homework = new Homework(1, tommorow, "Math", TaskType.Default, "Complete the math assignment");

// Act

bool result = homework.DoTask(tommorow);

// Assert

Assert.IsTrue(result);

Assert.IsTrue(homework.Done);

}

[TestMethod]

public void DoTask\_WithDifferentDeadline\_ReturnsFalse()

{

// Arrange

DateTime deadline = DateTime.Today.AddDays(1);

Homework homework = new Homework(1, deadline, "Math", TaskType.Default, "Complete the math assignment");

// Act

bool result = homework.DoTask(DateTime.Today);

// Assert

Assert.IsFalse(result);

Assert.IsFalse(homework.Done);

}

[TestMethod]

public void Parse\_ValidInput\_ParsesHomework()

{

// Arrange

string input = "1|12.11.23|Math|Write|Complete the math assignment";

// Act

Homework homework = Homework.Parse(input);

// Assert

Assert.AreEqual(1, homework.TaskNumber);

Assert.AreEqual(new DateTime(2023, 11, 12), homework.Deadline);

Assert.AreEqual("Math", homework.Subject);

Assert.AreEqual(TaskType.Write, homework.taskType);

Assert.AreEqual("Complete the math assignment", homework.taskText);

}

[TestMethod]

public void TryParse\_ValidInput\_ParsesHomework()

{

// Arrange

string expected = "1|12.11.23|Math|Default|Complete the math assignment";

Homework actual;

Exception error;

// Act

bool result = Homework.TryParse(expected, out actual, out error);

// Assert

Assert.IsTrue(result);

Assert.IsNotNull(actual);

Assert.AreEqual(1, actual.TaskNumber);

Assert.AreEqual(new DateTime(2023, 11, 12), actual.Deadline);

Assert.AreEqual("Math", actual.Subject);

Assert.AreEqual(TaskType.Default, actual.taskType);

Assert.AreEqual("Complete the math assignment", actual.taskText);

}

[TestMethod]

public void ToString\_ReturnsFormattedString()

{

// Arrange

Homework homework = new Homework(1, new DateTime(2023, 11, 12), "Math", TaskType.Default, "Complete the math assignment");

// Act

string result = homework.ToString();

// Assert

string expected = "Task Number: 1 | Deadline: 12.11.23 | Subject: Math | Task Type: Default | Task Text: Complete the math assignment";

Assert.AreEqual(expected, result);

}

[TestMethod]

public void GetStatistics\_CalculatesStatistics\_NoINcomplete()

{

Storage.GetTasks().Clear();

int d = Storage.GetTasks().Count;

int b = Homework.ObjectCount;

// Arrange

List<Homework> db = new List<Homework>

{

new Homework(111, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment"),

new Homework(222, DateTime.Today.AddDays(-2), "Science", TaskType.Default, "Complete the science assignment"),

new Homework(333, DateTime.Today.AddDays(-4), "History", TaskType.Default, "Complete the history assignment")

};

foreach (var t in db)

{

t.DoTask(t.TaskNumber);

}

// Act

Homework.GetStatistics(Storage.GetTasks());

// Assert

Assert.AreEqual(3, Homework.TotalTasks);

Assert.AreEqual(3, Homework.CompletedTasks);

Assert.AreEqual(0, Homework.IncompleteTasks);

Assert.AreEqual(TimeSpan.FromDays(2), Homework.AverageCompletionTime);

}

[TestMethod]

public void GetStatistics\_CalculatesStatistics\_ONEcompleted()

{

Storage.GetTasks().Clear();

// Arrange

List<Homework> db = new List<Homework>

{

new Homework(111, DateTime.Today, "Math", TaskType.Default, "Complete the math assignment"),

new Homework(222, DateTime.Today.AddDays(-2), "Science", TaskType.Default, "Complete the science assignment"),

new Homework(333, DateTime.Today.AddDays(-4), "History", TaskType.Default, "Complete the history assignment")

};

db[2].DoTask(db[2].TaskNumber);

// Act

Homework.GetStatistics(Storage.GetTasks());

// Assert

Assert.AreEqual(3, Homework.TotalTasks);

Assert.AreEqual(1, Homework.CompletedTasks);

Assert.AreEqual(2, Homework.IncompleteTasks);

Assert.AreEqual(TimeSpan.FromDays(4), Homework.AverageCompletionTime);

}

}

}