**THE MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE**

**State University of Intelligent Technologies and Communications**

**Department of Software Engineering**

**COURSE PROJECT**

**from the discipline «Object oriented programming»**

**On the theme:**

**«Educational application for higher education institutions»**

­of student of 2nd year of group ІПЗ ТЕ 2.1.01

of specialty 121 «Software Engineering»  
Kadian Richard Георгійовича

Керівник: ст. викл. кафедри ІТ   
к.і.н. Прокоп Ю.В.

Національна шкала

Кількість балів \_\_\_\_\_

Члени комісії:

Оцінка ECTS

підпис прізвище та ініціали

підпис прізвище та ініціали

Odesa – 2021 р.

**State University of Intelligent Technologies and Communications**

Навчально-науковий інститут інфокомунікацій та програмної інженерії

Department of Software Engineering

# Task

FOR COURSE PROJECT

From the discipline «Object oriented programming»

Of student Kadian Richard Георгійовича

Of specialty 121 «Software Engineering»

Of second year of group ІПЗ ТЕ-1.1.01

On the theme: «Educational application for higher education institutions»

Variant: 8

Вихідні дані:

Текствові файл F1 формату txt із заданою матрицею 10х10.

Текстовий файл F2 з текстом на 10 рядків.

Курсова робота виконується в наступному об’ємі:

1. Розрахунково-пояснювальна записка.

Теоритичні відомості про мову програмування С++.

Опис функції програми.

Опис функції бібліотеки KadianLibrary.

1. Графічни частина.

Блок схеми функцій, які були використані для виконань завдань.

# CALENDAR PLAN

|  |  |  |
| --- | --- | --- |
| № | Завдання | Термін виконання |
| 1 | Ознайомлення з завданням до курсового проєкту. Підготування вихідних файлів | 26.10.2020-30.10.2020 |
| 2 | Частина 1. Опрацювання матриць | 31.10.2020-06.11.2020 |
| 3 | Частина 2. Опрацювання динамічних масивів | 07.11.2020-13.11.2020 |
| 4 | Частина 3. Опрацювання рядкових даних | 14.11.2020-20.11.2020 |
| 5 | Частина 4. Опрацювання текстових файлів | 21.11.2020-27.11.2020 |
| 5 | Writing documentation | 18.11.2021-02.12.2021 |

# Table of contents

[1 Task 2](#_Toc89028635)

[2 CALENDAR PLAN 3](#_Toc89028636)

[3 Table of contents 4](#_Toc89028637)

[4 Introduction 6](#_Toc89028638)

[5 FORMATION OF REQUIREMENTS 7](#_Toc89028639)

[5.1 General requirements 7](#_Toc89028640)

[5.2 Analysis of The Development Environment 13](#_Toc89028641)

[5.3 Object Oriented Analysis 14](#_Toc89028642)

[6 Definition of data and methods 22](#_Toc89028643)

[7 Structures of classes 26](#_Toc89028644)

[7.1 Interface IUser 26](#_Toc89028645)

[7.2 Class Teacher 28](#_Toc89028646)

[7.3 Class Student 29](#_Toc89028647)

[7.4 Interface IGroup 30](#_Toc89028648)

[7.5 Class Group 31](#_Toc89028649)

[7.6 Interface ILecture 32](#_Toc89028650)

[7.7 Class Lecture 33](#_Toc89028651)

[7.8 Interface ITest 34](#_Toc89028652)

[7.9 Class Test 35](#_Toc89028653)

[7.10 Interface ITestQuestion 36](#_Toc89028654)

[7.11 Class TestQuestion 37](#_Toc89028655)

[7.12 Interface ITestMark 38](#_Toc89028656)

[7.13 Class TestMark 39](#_Toc89028657)

[7.14 Class Rules 40](#_Toc89028658)

[7.15 Class Services 41](#_Toc89028659)

[7.16 Class RegistrationServices 42](#_Toc89028660)

[7.17 Form Form1 43](#_Toc89028661)

[7.18 Form Registration 44](#_Toc89028662)

[7.19 Form ForgotPass 45](#_Toc89028663)

[7.20 Form TeacherMainMenu 46](#_Toc89028664)

[7.21 Form GroupInfo 47](#_Toc89028665)

[7.22 Form Add\_Lecture 48](#_Toc89028666)

[7.23 Form AddTest 49](#_Toc89028667)

[7.24 Form Add\_Test\_Question 50](#_Toc89028668)

[7.25 Form StudentStatistics 51](#_Toc89028669)

[7.26 Form StudentMainMenu 52](#_Toc89028670)

[7.27 Form SubjectTasksStudent 53](#_Toc89028671)

[7.28 Form ViewLecture 54](#_Toc89028672)

[7.29 Form ViewTest 55](#_Toc89028673)

[8 Structures of files 56](#_Toc89028674)

[9 Instruction For users 57](#_Toc89028675)

[10 List of used literature and sources 58](#_Toc89028676)

[11 Appendix А. Code of classes 59](#_Toc89028677)

[12 Appendix B. Class diagram 60](#_Toc89028678)

# Introduction

The purpose of the course work is to deepen and consolidate the knowledge gained during the study of the discipline "Object-Oriented Programming", and to acquire practical skills in designing and debugging programs that use classes and objects.

The student must develop a curriculum.

The training program must perform the following functions:

- student registration;

- presentation of material for training;

- ensuring control of knowledge;

- introduction and adjustment of educational material;

- obtaining information about the student's success.

# FORMATION OF REQUIREMENTS

## General requirements

1. Загальні положення

Метою курсової роботи є поглиблення та закріплення знань, одержаних при вивченні дисципліни «Об’єктно-орієнтоване програмування», та набуття практичних навичок у проектуванні та налагодженні програм, що застосовують класи та об’єкти.

Студент має розробити навчальну програму.

Навчальна програма повинна виконувати наступні функції:

- реєстрацію студента;

- представлення матеріалу до навчання;

- забезпечення контролю знань;

- введення та корегування навчального матеріалу;

- отримання інформації щодо успіхів студента.

1. Складові частини навчальної програми

Навчальна програма повинна складатися з наступних класів

1. Клас представлення навчального матеріалу

Клас обов’язково реалізує наступні функції.

Читання навчального матеріалу з файла.

Надання студенту порції навчального матеріалу.

Перехід до наступної порції матеріалу.

Клас додатково може реалізувати наступні функції відповідно до конкретного

завдання до курсової роботи.

1.1.Жорстка, заздалегідь встановлена схема подання матеріалу.

1.2.Матеріал подається за вибором студента.

1.3. Матеріал подається за вибором, але враховуючи зв’язок тем.

1.4. Матеріал подається за схемою, що задається викладачем для кожного

конкретного випадку.

1. Клас контролю знань студента

Клас обов’язково реалізує наступні функції.

Читання завдання до контролю з файлу.

Читання еталонних відповідей з файлу.

Надання студенту завдань до контролю знань.

Оцінювання кожної відповіді.

Клас додатково може реалізувати наступні функції відповідно до конкретного

завдання до курсової роботи.

2.1. Завдання типу «вибір альтернативної відповіді».

2.2. Завдання типу «вставити необхідне».

2.3. Завдання типу «отримати значення».

2.4. Завдання типу «вказати місце помилки».

2.5. Контроль повинен відбуватися наприкінці кожної порції.

2.6.Контроль повинен відбуватися наприкінці кожного розділу.

2.7.Контроль повинен відбуватися наприкінці кожного сеансу навчання.

2.8.Контроль повинен відбуватися наприкінці курсу навчання.

2.9.Встановлюється термін підготовки відповіді.

2.10.Фіксується час підготовки відповіді.

1. Клас реєстрації студента та видачі підсумкового документу

Клас обов’язково реалізує наступні функції.

Приймає дані щодо студента.

Видає підсумковий документ.

Клас додатково може реалізувати наступні функції відповідно до конкретного

завдання до курсової роботи.

3.1.Список студентів введено заздалегідь. Студент відшукує своє прізвище, читає

пароль, після цього пароль не можна прочитати.

3.2.Кожен студент повинен пройти реєстрацію сам. Він встановлює собі пароль.

3.3.Підсумковий документ містить тільки загальну оцінку.

3.4. Підсумковий документ містить усю історію навчання.

3.5. Підсумковий документ містить усі оцінки по порціям.

3.6. Підсумковий документ містить загальний час навчання.

3.7. Підсумковий документ містить назви тем, по яким студент мав незадовільні

оцінки.

1. Клас введення учбового матеріалу, контрольних завдань, еталонів відповідей та іншої інформації

Клас обов’язково реалізує наступні функції.

Введення учбового матеріалу у форматі, що задано іншими пунктами завдання.

Введення контрольних завдань у форматі, що задано іншими пунктами завдання.

Введення еталонів відповідей у форматі, що задано іншими пунктами завдання.

Клас додатково може реалізувати наступні функції відповідно до конкретного

завдання до курсової роботи.

4.1. Клас приймає дані тільки у діалоговому режимі.

4.2. Клас може приймати дані з іншого носія, що не входить до системи.

4.3. Клас дозволяє коригувати дані після закінчення сеансу їх введення, якщо

представлено пароль.

4.4. Клас дозволяє переглядати контрольні завдання та еталони відповідей, якщо

представлено пароль.

4.5. Клас дозволяє ввести список студентів, що мають вивчати даний предмет.

1. Клас контролю за ходом навчального процесу викладачем

Клас обов’язково реалізує наступні функції.

Перегляд результатів навчання для кожного студента.

Клас додатково може реалізувати наступні функції відповідно до конкретного

завдання до курсової роботи.

5.1.Отримання даних щодо загального часу навчання конкретного студента.

5.2.Отримання даних щодо кількості сеансів навчання для конкретного студента.

5.3.Отримання даних щодо оцінок, які виставлено системою для конкретного

студента.

5.4.Отримання даних щодо тем, які успішно пройшов конкретний студент.

Завдання на курсову роботу видається на початку 3 семестру.

Докладний зміст курсової роботи характеризується типовим завданням.

1. Зміст та порядок виконання курсової роботи

1. Постановка задачі.

1.1 Уточнення вимог до програмного продукту.

1.2 Об’єктно-орієнтований аналіз, визначення об’єктів і класів.

1.3 Визначення даних і методів.

2. Проектування класів та написання коду.

2.1 Проектування класів.

2.2 Написання коду у середовищі Visual C#.

3. Створення інтерфейсу користувача та файлів даних.

3.1 Створення графічних елементів керування.

3.2 Створення засобів обробки подій.

3.3 Створення файлів даних.

4. Перевірка функціонування програмного продукту.

Пояснювальна записка повинна включати:

- завдання до курсової роботи;

- анотацію;

- зміст;

- формулювання вимог;

- структури класів;

- коди класів;

- структури файлів;

- інструкцію користувача та контрольний приклад;

- результати тестування.

Курсова робота подається на перевірку не пізніше 11 тижня.

Захист курсової роботи проводиться в комісії на 12 тижні.

1. Індивідуальні завдання

Варіант No1.

Учбовий матеріал – елементарні конструкції мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.1, 2.2, 3.2, 4.2, 5.1

Варіант No2.

Учбовий матеріал – оператори розгалужень мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.2, 2.1, 3.1, 4.1, 5.2

Варіант No3.

Учбовий матеріал – оператори циклу мови С++.

Треба реалізувати наступні вимоги до програмних класів: 1.3, 2.3, 3.3, 4.2, 5.3

Варіант No4.

Учбовий матеріал – масиви мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.4, 2.4, 3.4, 4.3, 5.4

Варіант No5.

Учбовий матеріал – рядки символів мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.3, 2.5, 3.5, 4.1, 5.3

Варіант No6.

Учбовий матеріал – структури мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.1, 2.6, 3.6, 4.2, 5.2

Варіант No7.

Учбовий матеріал – функції з параметрами базових типів мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.2, 2.7, 3.7, 4.3, 5.1

Варіант No8.

Учбовий матеріал – функції з параметрами-масивами мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.4, 2.1, 3.6, 4.1, 5.4

Варіант No9.

Учбовий матеріал – функції з параметрами-рядками символів мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.3, 2.3, 3.1, 4.4, 5.3

Варіант No10.

Учбовий матеріал – функції з параметрами-структурами мови С++.

Треба реалізувати наступні вимоги до програмних класів:1.1, 2.2, 3.5, 4.2, 5.1

Варіант No11.

Учбовий матеріал – визначення класу у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.2, 2.5, 3.4, 4.1, 5.2

Варіант No12.

Учбовий матеріал – типи конструкторів у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.4, 2.9, 3.7, 4.3, 5.4

Варіант No13.

Учбовий матеріал – деструктори та динамічні об’єкти у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.3, 2.10, 3.6, 4.2, 5.3

Варіант No14.

Учбовий матеріал – дружні функції у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.1, 2.8, 3.5, 4.1, 5.4

Варіант No15.

Учбовий матеріал – віртуальні функції у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.2, 2.3,3.2, 4.4, 5.1

Варіант No16.

Учбовий матеріал – перевизначення операцій у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.4, 2.5, 3.1, 4.3, 5.2

Варіант No17.

Учбовий матеріал – спадкування у мові С++.

Треба реалізувати наступні вимоги до програмних класів:1.3, 2.4, 3.3, 4.1, 5.3

## Analysis of the Development Environment

Microsoft Visual Studio is a series of Microsoft products that include an integrated software development environment and a number of other tools. These products allow you to develop both console and GUI applications, including support for Windows Forms technology.

C # is an object-oriented programming language with a secure typing system for the .NET platform.

The syntax of C # is close to C ++ and Java. The language has a strict static typing, supports polymorphism, operator overload, pointers to member functions of classes, attributes, events, properties, exceptions, comments in XML format. Adopting many of its predecessors - C ++, Object Pascal, Module and Smalltalk - C #, based on the practice of their use, excludes some models that have proven to be problematic in software development, such as C #, in contrast to C ++, does not involve multiple inheritance of classes.

## Object Oriented Analysis

* According to Formation of Requirements, point 3)

@startuml

title Scenarios of using a lecture

:Teacher: as teacher

:Student: as student

(Lection) as lesson

(Add) as add

(Edit) as edit

(View) as view

student --> view

teacher --> add

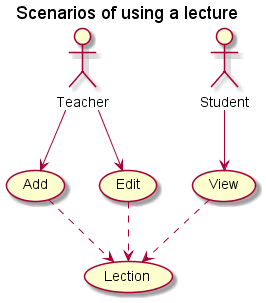
teacher --> edit

add ..> lesson

edit ..> lesson

view ..> lesson

@enduml



Also, according to the 8th variant, requirement 1.4 (the material is presented according to the scheme set by the teacher for each case) must be implemented. It means that the realization of the feature of reordering of the lectures must be fulfilled.

@startuml

title Editing list of Lectures

:Teacher: as teacher

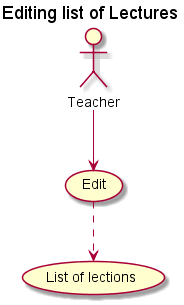
(List of lections) as lection

(Edit) as edit

teacher --> edit

edit ..> lection

@enduml



* According to Formation of Requirements, point 4)

@startuml

title Scenarios of using a Test

:Teacher: as teacher

:Student: as student

(Test) as lesson

(Add) as add

(Edit) as edit

(View) as view

student --> view

teacher --> add

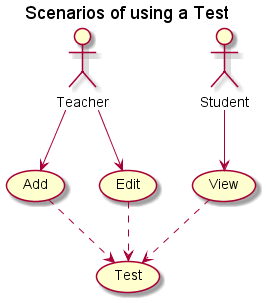
teacher --> edit

add ..> lesson

edit ..> lesson

view ..> lesson

@enduml



According to the 8th variant, requirement 2.1 (tasks such as "choosing an alternative answer") must be implemented. It means that the realization of the possibility to add a question with an open answer must be fulfilled.

@startuml

title Types of questions

(Question) as question

(One right answer) as one

(Multiple right answers) as mult

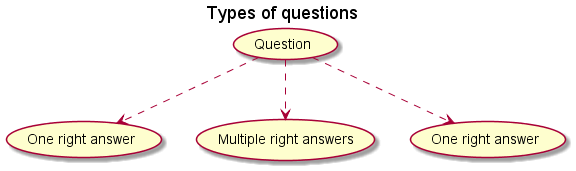
(One right answer) as alt

question ..> one

question ..> mult

question ..> alt

@enduml



Adding this to the use case diagram for test, we obtain:

@startuml

title Test use case from teacher side

:Teacher: as teacher

(Test) as test

(Question) as question

(Add) as add

(Edit) as edit

(Add) as addQ

(Edit) as editQ

teacher --> add

teacher --> edit

add ..> test

edit ..> test

test ..> addQ

test ..> editQ

addQ ..> question

editQ ..> question

(One right answer) as one

(Multiple right answers) as mult

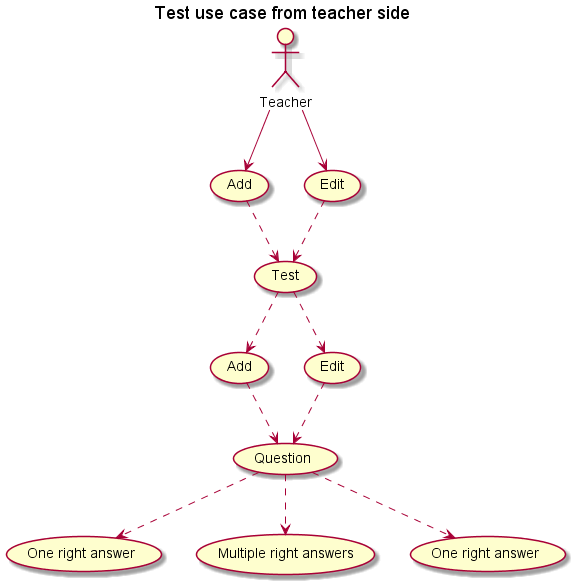
(One right answer) as alt

question ..> one

question ..> mult

question ..> alt

@enduml



* According to Formation of Requirements, point 5)

@startuml

title Student control system use cases

:Teacher: as teacher

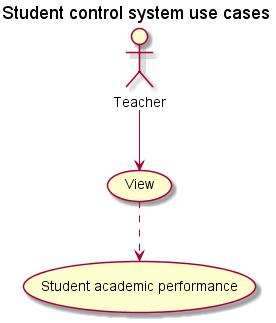
(View) as view

(Student academic performance) as sap

teacher --> view

view ..> sap

@enduml



According to the 8th variant, requirement 3.6 (the final document contains the total study time) must be implemented. It means that the realization of the possibility for the teacher to check the total study time for a certain student must be fulfilled.

@startuml

title Student control system use cases

:Teacher: as teacher

(View) as view

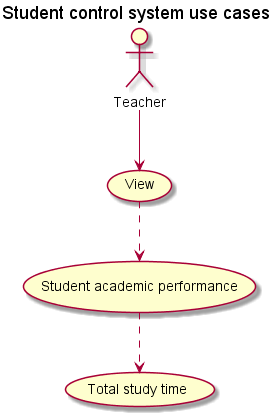
(Student academic performance) as sap

teacher --> view

view ..> sap

sap ..> (Total study time)

@enduml



* According to Formation of Requirements, point 6) and to the 8th variant, requirement 4.1(classes that add educational material, control tasks, standards of answers and other information accept data only in dialog mode)

It means that the only possibility for the teacher to add or edit lectures and tests is to add it using windows forms interface.

@startuml

title Lecture user interface

:Teacher: as teacher

(View) as view

(Add) as add

(Lecture) as lesson

(Windows froms interface) as IWinForm

teacher --> view

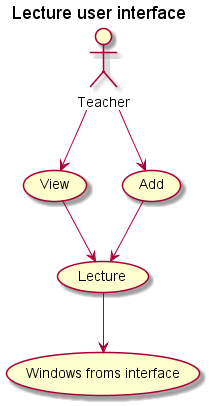
teacher --> add

add --> lesson

view --> lesson

lesson --> IWinForm

@enduml



@startuml

title Test user interface

:Teacher: as teacher

(View) as view

(Add) as add

(Test) as lesson

(Windows froms interface) as IWinForm

teacher --> view

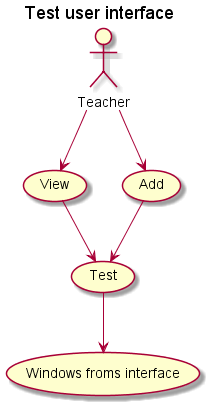
teacher --> add

add --> lesson

view --> lesson

lesson --> IWinForm

@enduml



* According to Formation of Requirements, point 7)

@startuml

title Abstract student control system use cases

:Student: as student

(View) as studentView

(Test) as test

(Lection) as lection

(Send result) as sr1

(Send result) as sr2

(Student academic performance) as sap

student --> studentView

studentView --> lection

studentView --> test

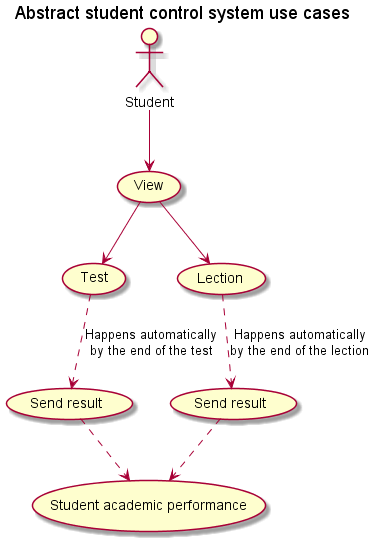
lection ..> sr1: "Happens automatically\nby the end of the lection"

test ..> sr2: "Happens automatically\nby the end of the test"

sr1 ..> sap

sr2 ..> sap

@enduml



According to the 8th variant, requirement 5.4 (Obtaining data on topics that a particular student has successfully passed. The assignment for the term paper is issued at the beginning of the 3rd semester. The detailed content of the course work is characterized by a typical task) must be implemented.

For obtaining data on topics that a particular student has successfully passed, marks of test must be collected. Also, according to the 8th variant requirement 3.6, the time spent on a lecture or a test must be collected as well. Thus we obtain such diagram on how data about student activity is collected:

@startuml

title Student data collection use cases

:Student: as student

(View) as studentView

(Test) as test

(Lection) as lection

(Send time spent) as sts

(Send mark) as sm

(Student academic performance) as sap

student --> studentView

studentView --> lection

studentView --> test

lection ..> sts: "Happens automatically\nby the end of the lection"

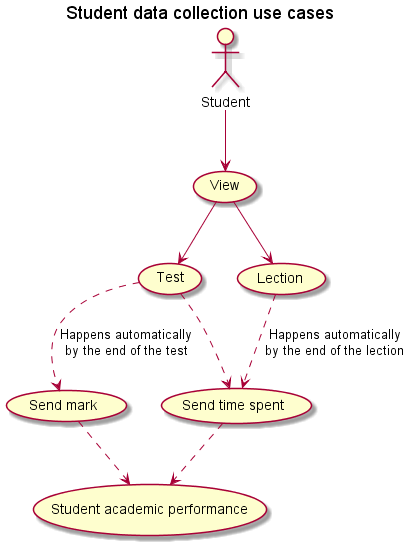
test ..> sts

test ..> sm: "Happens automatically\nby the end of the test"

sm ..> sap

sts ..> sap

@enduml



The next diagram presents the way how teacher should be able to view the performance of a particular student:

@startuml

title Teacher inspection of student performance

:Teacher: as teacher

(View) as view

(Student academic performance) as sap

teacher --> view

view --> sap

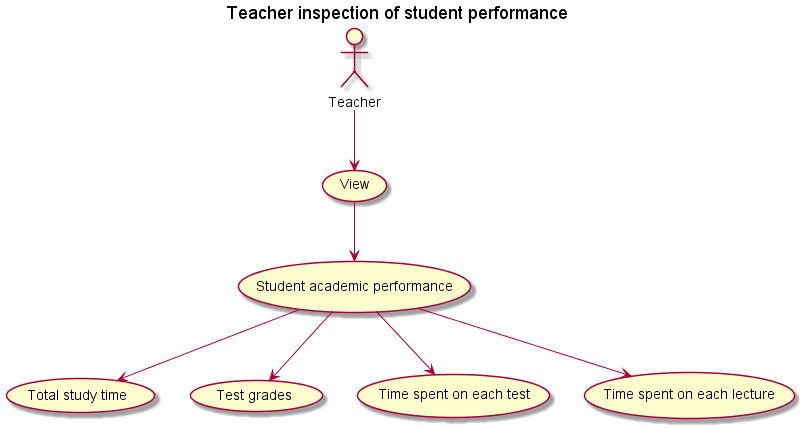
sap --> (Total study time)

sap --> (Test grades)

sap --> (Time spent on each test)

sap --> (Time spent on each lecture)

@enduml



# Definition of data and methods

Now that the object oriented analysis of the requirements has been done, the diagram of the relationships of classes can be done.

First, let’s create a class relationship of users. In this project there are two types of users: teacher and student. It is logical to assume that both teacher and student have some common attributes, such as name. Later, more common attributes may occur. However, there are some attributes that are needed only for students or for teachers. For example, a student has to be related to some group that he or she studies in. A teacher definitely does not need such attribute, but a teacher must have some information about the subject that he or she leads. Thus, it is clear that both student and teacher must have some common attributes, but they do not share all of them. Now it is needed to describe the concepts of teacher and student using the object oriented approach.

In object oriented programming the main type of abstraction is a class. It is like a blueprint of a specific object in real world. Every real world object has some color, shape, and functionalities - for example, the luxury car Ferrari. Ferrari is an object of the luxury car type. The luxury car is a class that indicates some characteristics like speed, color, shape, interior, etc. So any company that makes a car that meets those requirements is an object of the luxury car type. For example, every single car of BMW, Lamborghini, and Cadillac are an object of the class called 'Luxury Car'. Here, 'Luxury Car' is a class, and every single physical car is an object of the luxury car class.

Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.

In the case of this course project, we have tцo real life objects: teacher and student. Consequently, we can create a class that unifies them. It’s not going to be called “Human” because it is overly abstract for this project. There will be no other types of humans, so the class will be called “User”. In the case of this course project, both teacher and student are going to be users of the educational program.

In the preceeding paragraph the term “abstract” was used. It is one of the main paradigms of the object oriented programming. There are four of them. Understanding of each of them is crucially important for understanding the essense of the object oriented approach to programming. Below are given the definitions of each of them:

1. Abstraction

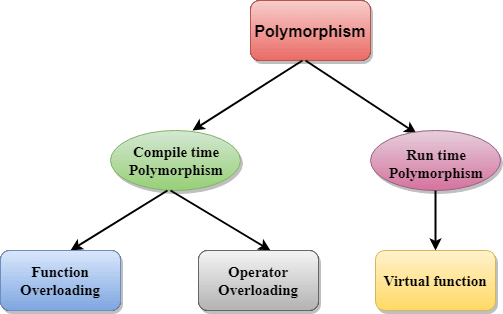
It is the property by virtue of which only the essential details are exhibited to the user. The trivial or the non-essentials units aren’t exhibited to the user.

Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details (this is what was done when the concept of “User” was preffered over the concept of “Human”). The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying or grouping the objects.

Example: Consider a real-life scenario of withdrawing money from ATM. The user only knows that in ATM machine first enter ATM card, then enter the pin code of ATM card, and then enter the amount which he/she wants to withdraw and at last, he/she gets their money. The user does not know about the inner mechanism of the ATM or the implementation of withdrawing money etc. The user just simply knows how to operate the ATM machine, this is called abstraction.

1. Polymorphism

Polymorphism means existing in many forms. There are two types of polymorphism which are run time polymorphism and compile-time polymorphism. Run time can take a different form while the application is running and compile-time can take a different form during compilation. All the unknown terms from this diagram will be explained later.



An excellent example of Polymorphism in Object-oriented programing is a cursor behavior. A cursor may take different forms like an arrow, a line, cross, or other shapes depending on the behavior of the user or the program mode. With polymorphism, a method or subclass can define its behaviors and attributes while retaining some of the functionality of its parent class. This means you can have a class that displays date and time, and then you can create a method to inherit the class but should display a welcome message alongside the date and time. The goals of Polymorphism in object oriented programming is to enforce simplicity, making codes more extendable and easily maintaining applications.

Inheritance allows you to create class hierarchies, where a base class gives its behavior and attributes to a derived class. One is free to modify or extend its functionality. Polymorphism ensures that the proper method will be executed based on the calling object’s type.

1. Inheritance

In C#, code is written in classes or blocks. Classes can interact with one another by using the properties of each block or extending the functionalities of a block through inheritance. Inheritance ensures that code is reused, but not rewriten. In terms of object oriented programming the concept of “reusability” is used when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

There are millions of libraries that a programmer can use through inheritance. The properties of a class can be inherited and extended by other classes. There are two types of classes. One is the parent or base class, and the other is the child class which can inherit the properties of the parent class. Inheritance is a major pillar in object oriented programming. It is the mechanism by which classes inherit attributes of other classes.

1. Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In a different way, encapsulation is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.

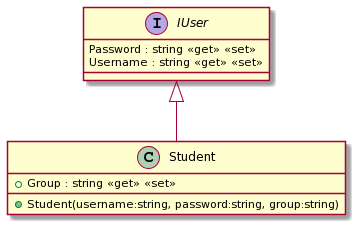
As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

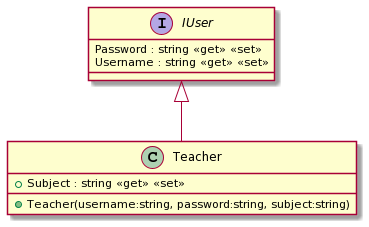
Encapsulation can be achieved by: Declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

However, there will be no need for an object such as User, there are only going to be teachers and students. Thus, an interface can be used.

An interface – maximal class abstraction in C#. It can contain declarations of methods, properties, indexers, and events. However, it cannot contain fields, and auto-implemented properties. As an example in the human world, a contract between the two or more humans binds them to act as per the contract. In the same way, an interface includes the declarations of related functionalities. The entities that implement the interface must provide the implementation of declared functionalities.

Therefore, classes teacher and student can implement functionality of interface User, in other words they will inherit from User. Here are two diagrams that represent these relationships.





# Structures of classes

## Interface IUser

This interface is used to declare the main fields for all users (teachers and stidents). The interface notation in C# looks like this:

interface Iuser { … }

Here, the word “interface” is the keyword to define the interface, and “IUser” is the name of the interface. It is considered a good manners to start an interface name with a big letter “I”.

An interface in C# is a type definition similar to a class, except that it purely represents a contract between an object and its user. It can neither be directly instantiated as an object, nor can data members be defined. So, an interface is nothing but a collection of method and property declarations.

Although a class can inherit from one class only, it can inherit from any number of interfaces. This is a simplified form of multiple inheritance supported by C#. When inheriting from a class and one or more interfaces, the base class should be provided first in the inheritance list, followed by any interfaces to be implemented. For example:

class MyClass : Class1, Interface1, Interface2 { … }

To charachterize the newly created interface, properties called fields can be used. A field of a class or interface is a variable of any type that is declared directly inside of it. A field is a property of some real life entity that charachterizes a class or interface that it is created in. This is how a propery is defined:

private string name;

Here, word “public” is an acces modifier. Access modifiers in C# are used to specify the scope of accessibility of a member of a class or type of the class itself. For example, a public class is accessible to everyone without any restrictions, while an internal class may be accessible to the assembly only. Access modifiers are an integral part of object-oriented programming. Access modifiers are used to implement encapsulation of OOP. Access modifiers allow you to define who does or who doesn't have access to certain features. In C#, there are four main access modifiers

1. Public – There are no restrictions on accessing public members.
2. Private – Access is limited to within the class definition. This is the default access modifier type if none is formally specified.
3. Protected – Access is limited to within the class definition and any class that inherits from the class.
4. Internal – Access is limited exclusively to classes defined within the current project assembly.

First three are the most popular. However, for fields the “private” access modifier is used the most because one of the main features of object orientated programming is information hiding and encapsulation. This means a class allows access to member variables only via an interface: getter and setter methods. So other classes cannot access the member variables and modify them in an unwanted way.

Getters and setters are methods used to declare or obtain the values of variables, usually private ones. They are important because it allows for a central location that is able to handle data prior to declaring it or returning it to the developer. Within a getter or setter you are able to consistently handle data that will eventually be passed into a variable or additional functions. An example of this would be a user’s name. If you are not using a setter and just declaring the userName field by hand you could end up with results as such: “kevin”, “KEVIN”, “KeViN”, “”, etc. With a setter you can not only adjust the value, for example, but you can also handle situations where the data is not valid such as the example where “” is passed. The same applies to a getter – when the data is being returned, you can modify the results for proper formatting further up the chain.

Therefore, the code will look loke this:

private string name;

public string GetName () {

return name;

}

public string SetName (string newName) {

// some validation if needed

name = newName;

}

However, C# allows to shorten the code above with automatic properties. An automatic property combines the field, getter, and setter into one line in the following manner:

public string Name { get; set; }

When you declare a property as shown in the example above, the compiler creates a private field that can only be accessed through the property's get and set accessors. So basically it is a syntax sugar that makes the code shorter and easier to read and faster to write.

Coming back to the description of IUser interface, all its field are written in the manner of automatic properties.

In the case of the current project, all users are going to have 4 fields:

1. Username
2. Password
3. Secret question
4. Secret answer

Username is a going to be unique for the whole program, and cannot be repeated by any user.

Password is going to be used by the user to enter its main menu.

Secret question is going to be used in the case if the forgets its password, and wants to restore it. This question is going to be outputted on the screen.

Secret answer is going to be used to check if the user answered correctly on the secret question. If the user answered correctly, he or she could set a new password.

## Class Teacher

This public class is used to store all the key information about a teacher. Because a techer is a user, it inherits from the IUser. However, it has additional information about itself to charachterize a teacher. In C#, for a class to inherit from an interface the following notation is used:

public class Teacher : Iuser { … }

As the class Teacher has to implement everything from the interface IUser, the automatic properties for fields from the interface must be written. However, there is ine additional field Subject. It refers to the subject that the teacher leads.

In this class, there is also a constructor used. A constructor is a special method of the class which gets automatically invoked whenever an instance of the class is created. Like methods, a constructor also contains the collection of instructions that are executed at the time of Object creation. It is used to assign initial values to the class foelds of the same class. Important thing about the name of constructors is that its name must be the same as the name of the class. For example, in the case of the class Teacher, a constructor will look in the following way:

public Teacher () { … }

Constructors usually have public acces modifier because they are used to initiate an instance of an object outside of the object. Usually, the parameters of a constructor are the values for the fields of the class it belongs to. For example, in the case of the class Teacher, a constructor that defines all the fields of it will look in the following way:

public Teacher(string username, string password, string secretQuestion, string secretAnswer, string subject) {

Password = password;

SecretAnswer = secretAnswer;

SecretQuestion = secretQuestion;

Subject = subject;

Username = username;

}

However, one class can implement as many constructors as needed, as long as their parameters are different.

## Class Student

This public class is used to store all the key information about a student. Because a student is also a user, it inherits from the IUser. However, it has additional information about itself to charachterize a student:

1. Group
2. StudyTime

Group is a field that contains the name of the group that the student studies in. An example of a name of a group looks like this: 121.1. Here, “121” is the specialty of that the student studies in, and “1” is the year year of studying of the student.

StudyTime is the total time that the student has spent studying. This field is not of a trivial type. Its type is TimeSpan. TimeSpan is a structure that represents a time interval. It is defined in the System namespace, so it is required to write the following line in the beginning of the file:

using System;

By using TimeSpan, several of the main features of the object oriented programming are used: incapsulation and abstraction. It is not needed to know all the details how TimeSpan works, how its fields are defined, how its methods work. All that is needed is the name of fields and methods, and general understanding of how they work, so basically what a method will return if it is called. To sum up, the details of the realization are hidden, they are encapsulated. And here, by using TimeSpan, it is used on some level of abstraction, because we only use the interface of TimeSpan, and the word “interface” in this case implies that only the names of the methods are known.

## Interface IGroup

This interface declares the main fields and methods for Groups. The fields are:

1. Name
2. Specialty
3. Year
4. Students
5. Subjects

Field Name has the same functionality as the field “Group” in the class Student.

Field Specialty defines the specialty of the group. It is obtained from the field Name.

Field Year defines the year of the group. It is obtained from the field Name.

Field Students stores a list of all students that belong to this group. It has a special type List that is defined in the System.Collections.Generic namespace. A List represents a strongly typed list of objects that can be accessed by index. It also provides methods to search, sort, and manipulate lists.

Field Subjects stores a list of all subject that a particular group has in the current semester.

The methods that this interface declares are:

1. addStudent – adds a student to the Students list.
2. rmStudent – removes a student from the Students list

## Class Group

This public class is used to store all the key information about a group.

The addStudent method implements the adding of a student to the Students field, but it also contains a checker that makes sure that this student has not already been added to the list.

The addStudent method implements the removal of a student from the Student field.

The Class froup also contains a constructor.

## Interface ILecture

This interface declares the main fields and methods for Lectures. The fields are:

1. ImgList
2. Name
3. Semester
4. Text
5. StudentMarksList

The ImgList is a list of paths to the images that are used in the current lecture.

Name is the title of the lecture.

Semester is the semester that the lecture must be shown in. It can be equal to 1 or 2,referring respectively to the first ans second semester.

Text is the main part of the lecture, the main text.

StudentMarksList collects information about all students that have read a lecture. In the case of the Lecture class, it is used to store information about the amount of time that students have spent on this lecture. This list has a special type TestMark. TestMark is a class of this project that will be described later.

This interface has declaration of the following methods:

1. addImg method must add a picture path to the ImgList list.
2. WriteToJson saves an instance of an object to the file.

## Class Lecture

This public class is used to store all the key information about a lecture. Because class Lecture inherits from the ILecture interface, it has to implement all the methods from the interface.

Class Lecture does not have any additional fields? Only thouse that it inherits.

It has two constructors. First is the default one: it creates the null-based values for all the fields. The other creates an instance of Lecture and sets values to all fields except for the StudentMarksList list.

Class Lecture implements the following methods from the interface:

addImg – adds the given filename (with absolute path) to the ImgList, but first check if the path empty.

WriteToJson – saves an instance of a Lecture object to a file, but firstly makes sure that the directory that was given exists.

## Interface ITest

This interface declares the main fields and methods for Test. The fields are:

1. RandQuestionOrder
2. Semester
3. StudentMarksList
4. Questions
5. Name

RandQuestionOrder is Boolean that defines whether the order of the question in the test must be randomized or shown in the order that the teacher has set.

Semester is the semester that the lecture must be shown in. Kust like in the case with Lecture, it can be equal to 1 or 2.

StudentMarksList has a broader usage compared to the Lecture class. It stores not only the time that students have spent to pass this test, but also what marks did they get for each question.

Questions stores a list of all question that this test contains. It has a special type TestQuestion that was created specifically for this project. The description of this class will be considered later.

Name is the title of the test.

This interface has declaration of the following methods:

1. getStudentMark – gets a mark of a specific student by the given username.
2. maxScore – calculates the maximal score that can be obtained in this test.
3. WriteToJson – saves an instance of an object to the file.
4. getQuestions – gets the titles of each question in the Questions list.

## Class Test

This public class is used to store all the key information about a Test. It inherits from the ITest interface, does not have any additional fields, and has two constructors: one default, and one with all fields, except for the StudentMarksList.

Class Test implements the following methods from the interface:

getStudentMark.

maxScore – calculates the maximal score that can be obtained in this test by summing up the values of each question in the Questions list.

WriteToJson – saves an instance of an object to the file, but checks if the given directory ends with a “/”, and that this directory exists.

getQuestions.

## Interface ITestQuestion

This interface declares the main fields and methods for TestQuestion. The fields are:

1. Value
2. RightAns
3. WrongAns
4. Question

Value of a question has a value from 1 to 4. It refers to the importance of a question relative to other question.

RightAns is a list that stores all the right answers that the question has.

WrongAns is a list that stores all the wrong answers that the question has.

Question is the actual question that the student must answer.

It is important to note that if there are no wrong answers, then it is a question with an alternative answer. This is how, according to variant 8, the functionality of a question with an alternative answer will be implemented. If there is only one right answer, then it is a question that has obly one right answer. If the question has moe than one right answer, then it is a multiple right answers question.

## Class TestQuestion

This public class is used to store all the key information about a test question. It inherits from the ITestQuestion interface, does not have any additional fields or methods, and it does not have a constructor.

## Interface ITestMark

This interface declares the main fields and methods for TestMark. The fields are:

1. Marks
2. StudentUsrName
3. TimeSpent

Marks is a list of marks that the student received for answering each of the questions.

StudentUsrName is the username of the student that has received the according marks in the Marks list.

TimeSpent has TimeSpan type, and it stores the amout of time that the student has spent on a test.

This interface also contains a declaration of the following method:

getStudentMark – calculates the total mark for the whole test of the student in a 100 degree scale.

## Class TestMark

This public class is used to store all the key information about a test mark. It inherits from the ITestMark interface, does not have any additional fields or methods, and has two constructors: one default, and one that set values to all the fields of TestMark class.

Class Test implements the following method from the interface:

getStudentMark – returns a mark of the student on a 100 degree scale by a formula.

## Class Rules

This public class is used to store information about the list of subject that a particular specialty has in a particular year and semester. It class is used only to get information from it. It has four fields:

1. S121
2. S122
3. S123
4. S172

All of these fields represent different specialties. Each of these fields is a two-dimensional array 4 by 2. There are four years of studying, and each year consists of two semesters.

There is noother way to change the fields, except to change the vode of the Rules class. Therefore, they are only for reading.

The Rules Class has a method getSubjList. It gets a list of subjects based on the specialty and the year that is given. And as it is already known, specialty and year form the group name. Thus, there might be a situation, when the name of the group is known, but not separately the specialty and year. In such a case, important feature of the object oriented programming can be used: polymorphism. Here, one method is needed to handle the getting of subject list, but there might be two variants of the incoming data: when group name is known, and when the specialty and the year is known. Even though the incoming parameters are different, the general algorithm is similar, nearly the same.

To avoid creation of two functions with different names, two functions can be created with the same name and returning type, but different arguments. This type of polymorphism is called function overloading. The following code shows how the polymorphism is used in the Rules class:

// two arguments

public static string[] getSubjList(string spec, string year) { … }

// one argument

public static string[] getSubjList(string grName) {

Since this class is used only to read from its fields, and it is never going to be modified, all the fields and methods have a static modifier. The static modifier is used to declare a static member, which belongs to the type itself rather than to a specific instance of a class.

## Class Services

This public class is used to create some level of abstraction between forms and algorithms that process user activity. It encapsulates many methods to make the code of forms readable, shorter, and lighter by taking out some of the algorithms outside of them. Thus, only a call of a method from the Services class is needed to perform the algorithm. Subsequently, the code of a form becomes more readable.

The total number or references to the Services class is 64, and the number of references for a single method get up to 8 times. This is called code reusability. It is a great strength of the object oriented programming. The code consists of many pieces of code that come together and create a program.

The Services class has the following methods:

1. getRand – takes two arguments: *integer* *start*, and *integer end*. Returns a random value between *start* and *end*.
2. GetCurrentSemester – takes no arguments. Returns the current semester based on the current date.
3. getOrder – takes one argument: *string dir*. Returns a list of lectures or tests from the directory *dir* that are stored in the file called order.txt. The filesystem and file names will be described in the section [8 FILE STRUCTURE](#_Structures_of_files).
4. rewriteOrder – takes two arguments: *string dir*, *List<string> newOrder*. Creates a new file or rewrites the existing one with the order from *newOrder*.
5. appendToOrder – takes two arguments: *string dir*, *string nameToAppend*. Appends to the file order.txt in the directory *dir* *nameToAppend*.
6. saveOrder – takes two arguments: *string dir*, *DataGridView dgv*. Saves the names of lectures of tests from a dataGridView *dgv* to file order.txt to directory *dir*.
7. btnOnClick – delegate function. A delegate is like a pointer to a function. It is a reference type data type and it holds the reference of a method. All the delegates are implicitly derived from System.Delegate class. A delegate can be declared using delegate keyword followed by a function signature as shown below.

public delegate void btnOnClick(object sender, EventArgs e);

This delegate can be used to point to any method that has same return type and parameters declared with it.

This function is going to be used as a type of argument for the function *createBtnList* that is also declared in the Services class.

1. createBtnList – takes four arguments: *int x, int y, List<string> btnTextList, btnOnClick delFunc*. Creates a list of buttons with the starting point (*x, y*), an onclick event *delFunc*, and names that are given in the *btnTextList*. Thus, the method will create as many buttons, as there are elements in the *btnTextList*.
2. createRadioBtnList –takes three arguments: *int x, int y, List<string> btnTextList*. Creates a list of radio buttons with the starting point (*x, y*), and names that are given in the *btnTextList*. The method will create as many radio buttons, as there are elements in the *btnTextList*.
3. createChkBoxList – takes three arguments: *int x, int y, List<string> btnTextList*. Creates a list of check boxes with the starting point (*x, y*), and names that are given in the *btnTextList*. The method will create as many check boxes, as there are elements in the *btnTextList*.
4. getGroupList – takes no arguments. Returns a list of all groups that have at least one student in them.
5. getGroupsWithSubj – takes one argument: *string subject*. Returns a list of groups that have subject *subject* this semester according to the class Rules.
6. deserializeObj – takes one argument: *string filePath*. Desirealizes a file that has an absolute name *filePath* from *.json* format to a class of type T. In this case, a feature of the C# called generic method is used. A generic method is a method that has a variable return type. This method is useful in cases when the user of the method might need to call this method for different types. The declaratio of such method looks in the following way:

public static T deserializeObj<T>(string filePath) { … }

In the case of this project, it is going to be necessary to desirealize objects of different types, so to shorten the code, one generic method can be written instead of several methods that do the save operations, but have different return types.

1. nextImg – takes three arguments: *ref int currentImg, Lecture lecture, PictureBox pcb*. It gets the next picture from the *lecture.ImgList*, and shows it to the *PictureBox pcb*. Additionally, it increments the index of the current picture *currentImg*, and to increment it not only in this method, but such that the caller knows the updated index, the *currentImg* is passed by reference.

This method is used to slide pictures in lectures.

1. previousImg – implements the same functionality as *nextImg* method, and has the same functionality, and instead decrements the *CurrentImg*.

This method is used to slide pictures in lectures.

1. randomizeList – takes one argument: *List<T> listToRandomize*. Has a generic type. Returns a list with the same element that are in *listToRandomize*, but in a randomized order.

It is used to randomize test questions if the test property RandQuestions requires it.

1. nextQuestion – it is used when when the student is in the progress of passing a test, and needs to switch to the next question.
2. getCollection – takes one argument: *Form Form*. Gets collection of type *T* and finds all the elements of this type on the form *Form*. Returns a list of all the elements that it has found.
3. derandomizeMarks – takes two arguments: *ViewTest vt, TimeSpan timeSpent*. It is used in a case when the test question are randomized, and do the marks are also randomized. Thus, it is needed to put them in the right order. This method is used to put marks in the correct order.
4. openCP – takes one argument: *GroupInfo groupInfo*. This method is used to show a form for the teacher to edit the task for the course project.
5. DGVCellContentClick – takes three arguments: *object sender, DataGridViewCellEventArgs e, int colInd*. It is used when a click on a button-cell in dataGridView happens to get a value from column with index *colInd*.
6. fillDGV – takes three arguments: *DataGridView dgw, List<string> nameList, string btnText*. It is used to fill a dataGridView that has three columns with the following headers: Index, Title, button Name.
7. DGVMouseClick – takes four arguments: *DataGridView dgv, MouseEventArgs e, ref DataGridViewRow Rw, ref int RowIndexFromMouseDown*. This method is used for reordering rows in dataGridView.
8. DGVDragEnter – takes two arguments: *DataGridView dgv, DragEventArgs e*. This method is used for reordering rows in dataGridView.
9. DGVragDrop – takes four arguments: *DataGridView dgv, DragEventArgs e, DataGridViewRow Rw, int RowIndexFromMouseDown*. This method is used for reordering rows in dataGridView.

## Class RegistrationServices

This public class is used to create some level of abstraction between forms and algorithms that process user activity as well. However, this has a very specific usage. It only concerns the validation during the registration.

It has only one field called regex. It is used only for reading and it has a type Regex. It is a system class in C#. Regular Expression is a pattern which is used to check whether the given input text is matching with the given pattern or not. C# provides a class termed as Regex which can be found in System.Text.RegularExpression namespace.

In the case of this project the following regular expression is used:

private static readonly Regex regex = new Regex("^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[#?!@$%^&\*-]).{8,}$");

This regular expression makes sure that the string contains at least:

1. 8 characters.
2. 1 capital letter.
3. 1 number.
4. 1 special symbol.

This class also has the following methods:

1. validateUsrName – takes one argument: *string usrName*. It is used on the stage of registration to check if the *usrName* mathches the requirements that the regular expression has. It is done using one of the methods of the Regex class called IsMatch(). This method also makes sure that this *usrName* has not been taken by some other user.
2. validatePass – takes one argument: *string password*. It is used on the stage of registration or resetting a password of a user to check if the password meets the requirements.
3. passMatch – takes two arguments: *string password*, *string repeatPass*. It is used on the stage of registration to check if the password that the user has created matches with its copy.

## Form Form1

This is the first form that is shown to the user when the program starts. It, and all the other forms in this project inherit from the standard C# Form class. This form has no fileds, but has a constructor that initializes the form, and the following methods:

1. button1\_Click – an event handler that is initiated when a standart event buttonOnClick happens as the user clicks the “Log in” button.

Events in C# are based on the delegate model. It enables a subscriber to register with and receive notifications from a provider. An event sender pushes a notification that an event has happened, and an event receiver receives that notification and defines a response to it. So, basically, an event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the event sender. The event sender doesn't know which object or method will receive (handle) the events it raises.

1. button2\_Click – an event handler that is initiated when the user clicks the “Registration” button. It opens the Registration Form.
2. button3\_Click – an event handler that is initiated when the user clicks the “Forgot password” button. It opens the ForgotPass Form.

User has three options after the the form is shown:

1. Login to his/her account.
2. Register a new account (either student account or teacher account).
3. Reset password if he/she forgot it.

The screenshots of all forms are presented in the Appendix C.

## Form Registration

It is used to register a new user, either a student or a teacher.

It has a field Form1 to store the link to the initiated Form1 in order to get back to it after the registration process was finished, and not to create anther instance of Form1. It helps to optimize the program. Storing of the Form1 is also needed because Form1 is hidden when Registration form starts. Form1 can not be closed because if it cloases, the whole program shuts down.

This form has one constructor that sets the value for the Form1 field.

Registration form has the following methods:

1. button1\_Click – an event handler that registers a user when he/she clicks the “Register” button.
2. button2\_Click – an event handler that closes the Registration form, and opens Form1 that is stored in the field Form1 when a user clicks the “Go back” button.
3. textBox1\_TextChanged – an event handler that hides error messages when the users starts to fix problems with invalid data in the fields.
4. textBox2\_TextChanged – an event handler that hides error messages when the users starts to fix problems with invalid data in the fields.
5. textBox3\_TextChanged – an event handler that hides error messages when the users starts to fix problems with invalid data in the fields.
6. radioButton1\_CheckedChanged – an event handler that happens when radioButton1 is checked. It enables fields that are specific for the teacher.
7. radioButton2\_CheckedChanged – an event handler that happens when radioButton2 is checked. It enables fields that are specific for the student.

## Form ForgotPass

## Form TeacherMainMenu

## Form GroupInfo

## Form Add\_Lecture

## Form AddTest

## Form Add\_Test\_Question

## Form StudentStatistics

## Form StudentMainMenu

## Form SubjectTasksStudent

## Form ViewLecture

## Form ViewTest

# Structures of files

# Instruction For users

1) For student

2) For teacher

No need for screenshots of the form. Just click here, click there, then here

# List of used literature and sources

1. <https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio>
2. <https://uk.wikipedia.org/wiki/C_Sharp>
3. <https://www.tutorialsteacher.com/csharp/csharp-class>
4. <https://www.geeksforgeeks.org/c-sharp-abstraction/?ref=lbp>
5. <https://www.nerd.vision/post/polymorphism-encapsulation-data-abstraction-and-inheritance-in-object-oriented-programming#:~:text=Abstraction%20ensures%20simplicity.,the%20attributes%20of%20another%20class.&text=Polymorphism%20allows%20program%20code%20to,be%20modified%20by%20external%20codes>.
6. <https://www.geeksforgeeks.org/c-sharp-encapsulation/?ref=lbp>
7. <https://www.tutorialsteacher.com/csharp/csharp-interface#:~:text=In%20C%23%2C%20an%20interface%20can,functionalities%20for%20the%20file%20operations>.
8. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/fields>
9. <https://www.c-sharpcorner.com/uploadfile/puranindia/what-are-access-modifiers-in-C-Sharp/#:~:text=Access%20modifiers%20in%20C%23%20are,accessible%20to%20the%20assembly%20only>.
10. <https://www.c-sharpcorner.com/interview-question/what-are-getters-and-setters-and-why-are-they-important>
11. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties>
12. <https://en.wikibooks.org/wiki/C_Sharp_Programming/Interfaces#:~:text=An%20INTERFACE%20in%20C%23%20is,of%20method%20and%20property%20declarations>.
13. <https://www.geeksforgeeks.org/c-sharp-constructors/#:~:text=A%20constructor%20is%20a%20special,the%20time%20of%20Object%20creation>.
14. <https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=net-6.0>
15. <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0>
16. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static>
17. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/generic-methods>
18. <https://www.geeksforgeeks.org/what-is-regular-expression-in-c-sharp/#:~:text=In%20C%23%2C%20Regular%20Expression%20is,generally%20termed%20as%20C%23%20Regex.&text=C%23%20provides%20a%20class%20termed,can%20be%20found%20in%20System>.
19. <https://docs.microsoft.com/en-us/dotnet/standard/events/#:~:text=To%20respond%20to%20an%20event,the%20user%20clicks%20a%20button>.

# Appendix А. Code of classes

File CourseProject.cpp:

1. #define \_CRT\_SECURE\_NO\_WARNINGS
2. #include <iostream>
3. #include <stdio.h>
4. #include <math.h>
5. #include <KadianLibrary.h>

# Appendix B. Class diagram

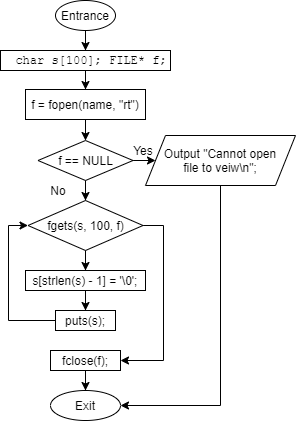


Рисунок Б.1 – функція viewFile.

# appendix c. Form screenshots