**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 a 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 year

**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 the second year of group ІПЗ ТЕ-1.1.01

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

Variant: 8

Initial data:

The course work is performed in the following volume:

1. Settlement and explanatory note.

Theoretical information about the C# programming language.

Description of classes of the program.

1. Graphical part.

Diagrams of classes.

Screenshots of use-cases.

# Calendar Plan

|  |  |  |
| --- | --- | --- |
| № | Task | Deadline |
| 1 | Acquaintance with the task to the course project | 16.09.2021-23.09.2021 |
| 2 | Part 1. Writing the registration, password restoration, and login forms | 23.09.2021-06.11.2021 |
| 3 | Part 2. Adding lectures | 28.102021-04.11.2021 |
| 4 | Part 3. Adding tests | 04.11.2021-11.11.2021 |
| 5 | Part 4. Viewing lectures and test | 11.11.2021-18.11.2021 |
| 6 | Part 5. Student tracking system | 18.11.2021-25.11.2021 |
| 7 | Writing documentation | 25.11.2021-02.12.2021 |

# Table of Contents

[1 Task 2](#_Toc89188898)

[2 Calendar Plan 3](#_Toc89188899)

[3 Table of Contents 4](#_Toc89188900)

[4 Introduction 6](#_Toc89188901)

[5 Formation of Requirements 7](#_Toc89188902)

[5.1 General requirements 7](#_Toc89188903)

[5.2 Analysis of the Development Environment 11](#_Toc89188904)

[5.3 Object-Oriented Analysis 12](#_Toc89188905)

[6 Definition of Data and Methods 20](#_Toc89188906)

[7 Structures of Classes 24](#_Toc89188907)

[7.1 Interface IUser 24](#_Toc89188908)

[7.2 Class Teacher 27](#_Toc89188909)

[7.3 Class Student 28](#_Toc89188910)

[7.4 Interface IGroup 29](#_Toc89188911)

[7.5 Class Group 30](#_Toc89188912)

[7.6 Interface ILecture 31](#_Toc89188913)

[7.7 Class Lecture 32](#_Toc89188914)

[7.8 Interface ITest 33](#_Toc89188915)

[7.9 Class Test 34](#_Toc89188916)

[7.10 Interface ITestQuestion 35](#_Toc89188917)

[7.11 Class TestQuestion 36](#_Toc89188918)

[7.12 Interface ITestMark 37](#_Toc89188919)

[7.13 Class TestMark 38](#_Toc89188920)

[7.14 Class Rules 39](#_Toc89188921)

[7.15 Class Services 40](#_Toc89188922)

[7.16 Class RegistrationServices 43](#_Toc89188923)

[7.17 Form Form1 44](#_Toc89188924)

[7.18 Form Registration 45](#_Toc89188925)

[7.19 Form ForgotPass 46](#_Toc89188926)

[7.20 Form TeacherMainMenu 47](#_Toc89188927)

[7.21 Form GroupInfo 48](#_Toc89188928)

[7.22 Form Add\_Lecture 49](#_Toc89188929)

[7.23 Form AddTest 50](#_Toc89188930)

[7.24 Form Add\_Test\_Question 51](#_Toc89188931)

[7.25 Form StudentStatistics 52](#_Toc89188932)

[7.26 Form StudentMainMenu 53](#_Toc89188933)

[7.27 Form SubjectTasksStudent 54](#_Toc89188934)

[7.28 Form ViewLecture 55](#_Toc89188935)

[7.29 Form ViewTest 56](#_Toc89188936)

[8 Structures of Files 57](#_Toc89188937)

[9 Instruction For Users 59](#_Toc89188938)

[10 List of Used Literature and Sources 60](#_Toc89188939)

[11 Appendix А. Code of Classes 62](#_Toc89188940)

[12 Appendix B. Class Diagram 121](#_Toc89188941)

[13 Appendix C. Form Screenshots 130](#_Toc89188942)

# 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

### General positions

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 correction of educational material;

- obtaining information about the student's success.

### Components of the curriculum

The curriculum should consist of the following classes.

### Class of presentation of educational material

The class must implement the following functions.

Reading educational material from a file.

Providing the student with a portion of educational material.

Go to the next portion of the material.

The class can additionally implement the following functions following the specific task of the course work.

1.1. Rigid, pre-established scheme of material presentation.

1.2. The material is submitted at the student's choice.

1.3. The material is optional, but takes into account the connection of topics.

1.4. The material is presented according to the scheme set by the teacher for each case.

### Student knowledge control class

The class must implement the following functions.

Read task to control from a file.

Read reference answers from a file.

Providing students with tasks for knowledge control.

Evaluation of each answer.

The class can additionally implement the following functions following the specific task of the course work.

2.1. Tasks such as "choosing an alternative answer".

2.2. Tasks such as "insert necessary".

2.3. Tasks such as "get value".

2.4. Tasks such as "specify the location of the error".

2.5. Control should occur at the end of each serving.

2.6. Control should take place at the end of each section.

2.7. Monitoring should take place at the end of each training session.

2.8. Control should take place at the end of the training course.

2.9. The term of preparation of the answer is established.

2.10. The time of preparation of the answer is fixed.

### Class of student registration and issuance of the final document

The class must implement the following functions.

Accepts student data.

Issues the final document.

The class can additionally implement the following functions according to the specific assignments for course work.

3.1. The list of students is entered in advance. The student finds his last name, reads the password, after which the password cannot be read.

3.2. Each student must register himself. He sets his password.

3.3. The final document contains only a general assessment.

3.4. The final document contains the entire history of training.

3.5. The final document contains all the estimates by portions.

3.6. The final document contains the total study time.

3.7. The final document contains the names of topics on which the student had unsatisfactory grades.

### Class of introduction of educational material, control tasks, standards of answers, and other information

The class must implement the following functions.

Introduction of educational material in the format specified by other items of the task.

Enter control tasks in the format specified by other task items.

Enter the standards of answers in the format specified by other items of the task.

The class can additionally implement the following functions following the specific task of the course work.

4.1. The class accepts data only in dialog mode.

4.2. The class can receive data from other media that are not part of the system.

4.3. The class allows you to adjust the data after the end of the session if a password is provided.

4.4. The class allows you to view control tasks and answer standards if a password is provided.

4.5. The class allows you to enter a list of students who must study this subject.

### Class control over the educational process by the teacher

The class must implement the following functions.

View learning outcomes for each student.

The class can additionally implement the following functions following the specific task of the course work.

5.1. Obtaining data on the total study time of a particular student.

5.2. Obtaining data on the number of study sessions for a particular student.

5.3. Obtaining data on the assessments made by the system for a specific

student.

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.

### Content and procedure for coursework

1. Statement of the problem.

1.1 Clarification of software product requirements.

1.2 Object-oriented analysis, the definition of objects and classes.

1.3 Definition of data and methods.

2. Designing classes and writing code.

2.1 Classroom design.

2.2 Writing code in Visual C #.

3. Creating a user interface and data files.

3.1 Creating graphical controls.

3.2 Creating event handling tools.

3.3 Creating data files.

4. Check the operation of the software product.

The explanatory note should include:

- tasks for course work;

- annotation;

- content;

- formulation of requirements;

- class structures;

- class codes;

- file structures;

- user manual and control example;

- test results.

The term paper is submitted for review no later than 11 weeks.

The course work is defended in the commission for 12 weeks.

### Individual variant

Variant №8

Training material – functions with parameters-arrays of C# language.

It is necessary to implement the following requirements for program classes: 1.4, 2.1, 3.6, 4.1, and 5.4.

## Analysis of the Development Environment

Microsoft Visual Studio is a series of Microsoft products that include an integrated software development environment and many 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 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 inheritances 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

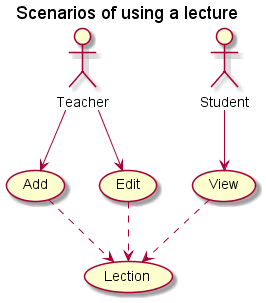
teacher --> edit

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

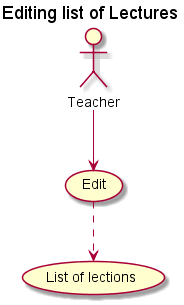
(List of lections) as lection

(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

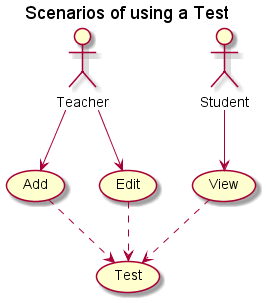
teacher --> edit

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

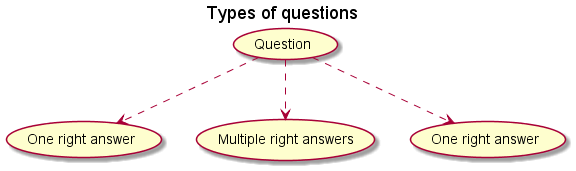
(One right answer) as alt

question ..> one

question ..> mult

question ..> alt

@enduml



Adding this to the use case diagram for a 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

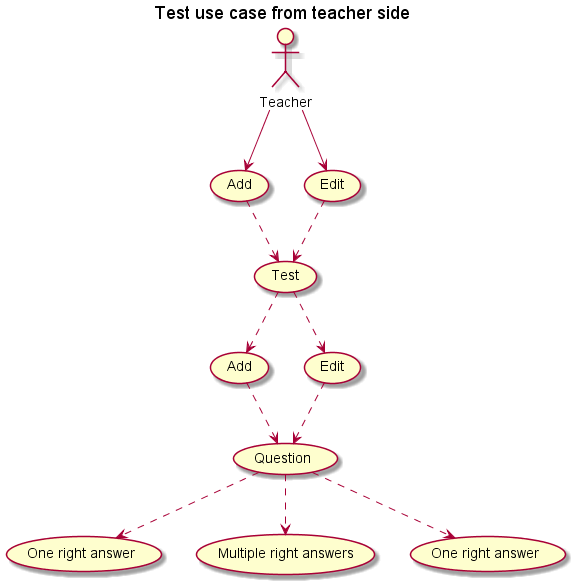
(One right answer) as alt

question ..> one

question ..> mult

question ..> alt

@enduml



* According to Formation of Requirements, point 5)

@startuml

title Student control system use cases

:Teacher: as teacher

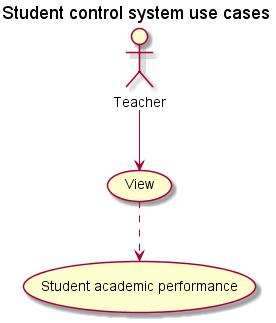
(View) as view

(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

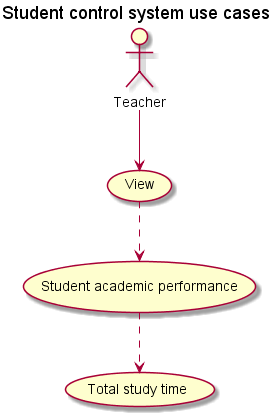
(Student academic performance) as sap

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 them using the 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

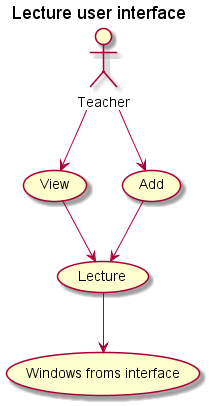
teacher --> add

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

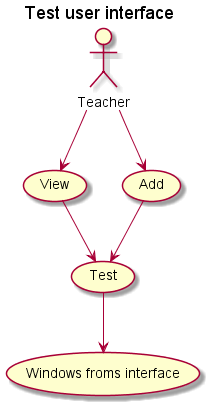
teacher --> add

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

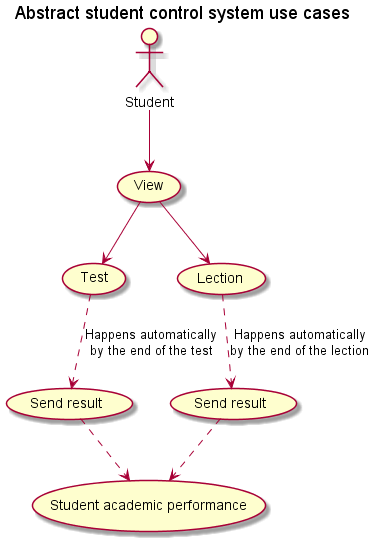
lection ..> sr1: "Happens automatically\nby the end of the lection"

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 the 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 a 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"

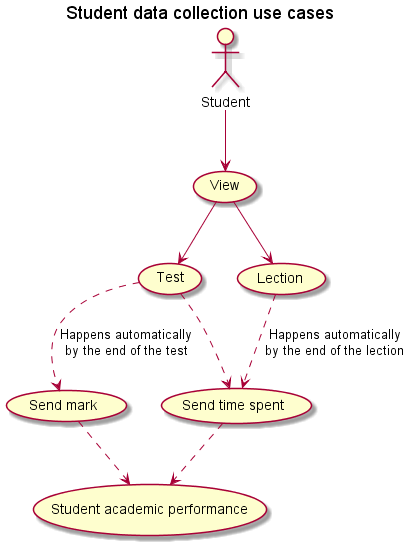
test ..> sts

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

sm ..> sap

sts ..> sap

@enduml



The next diagram presents the way how a 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

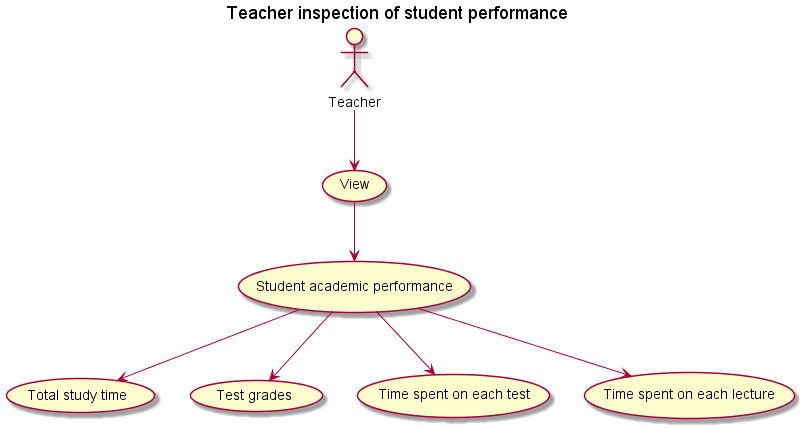
sap --> (Total study time)

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, some attributes 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 does not need such attributes, 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 the 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 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 preceding paragraph, the term “abstract” was used. It is one of the main paradigms of object-oriented programming. There are four of them. Understanding each of them is crucially important for understanding the essence of the object-oriented approach to programming. Below are given the definitions of each of them:

1. Abstraction

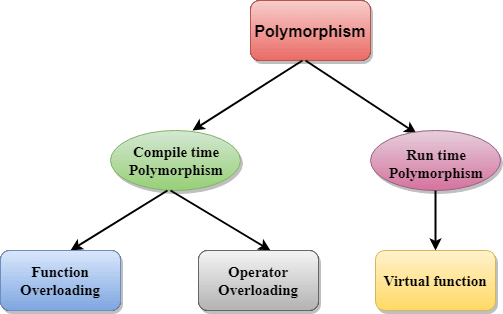
It is the property by which only the essential details are exhibited to the user. The trivial or the non-essentials units aren’t exhibited to the user.

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 preferred 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 an ATM. The user only knows that in ATM 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, 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 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 the 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 goal 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 rewritten. 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. Differently, 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 its class in which they are declared.

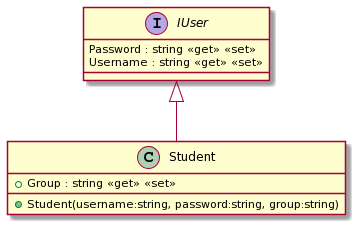
As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

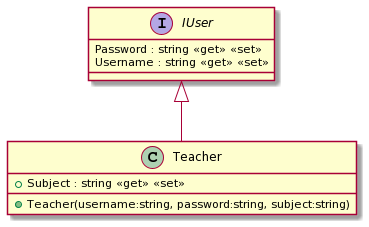
Encapsulation can be achieved by: Declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

However, there will be no need for an object such as a 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 students can implement the functionality of the interface User, in other words, they will inherit from the 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 students). 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 manner to start an interface name with the 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 inheritances 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 characterize 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 characterizes a class or interface that it is created in. This is how a property is defined:

private string name;

Here, the word “public” is an access modifier. Access modifiers in C# are used to specify the scope of accessibility of a member of a class or the 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.

The 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: a 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 can handle data before declaring it or returning it to the developer. Within a getter or setter, you can 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 like this:

private string name;

public string GetName () {

return name;

}

public string SetName (string newName) {

// some validation if needed

name = newName;

}

However, C# allows shortening 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 the IUser interface, all its fields 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

The Username is 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.

The 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 teacher is a user, it inherits from the IUser. However, it has additional information about itself to characterize 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 one 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 that 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 fields of the same class. An 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 access 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 characterize a student:

1. Group

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 that the student studies in, and “1” is the year of studying of the student.

## 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 of 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 subjects 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 Group 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 and 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 declarations 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 those 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 is empty.

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

## Interface ITest

This interface declares the main fields and methods for the Test class. 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. Just 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 questions 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 declarations 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 questions.

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 only one right answer. If the question has more 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 to marks in the Marks list.

TimeSpent has the TimeSpan type, and it stores the amount of time that the student has spent on a test. TimeSpent 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 at the beginning of the file:

using System;

By using TimeSpan, several of the main features of object-oriented programming are used: incapsulation and abstraction. It is not needed to know all the details of how TimeSpan works, how its fields are defined, how its methods work. All that is needed is the name of fields and methods, and a 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.

This interface also contains a declaration of the following method:

getStudentMark – calculates the total mark for the whole test of the student on 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 sets 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 subjects 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 study, and each year consists of two semesters.

There is no other way to change the fields, except to change the code 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, an important feature of object-oriented programming can be used: polymorphism. Here, one method is needed to handle the getting of the subject list, but there might be two variants of the incoming data: when the group name is known, and when the specialty and the year are known. Even though the incoming parameters are different, the general algorithm is similar, nearly the same.

To avoid the 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 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 of references to the Services class is 64, and the number of references for a single method gets up to 8 times. This is called code reusability. It is a great strength of 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 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 the 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 the 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 the 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 declaration 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 deserialize 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 elements 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 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 a 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 questions 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 the 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 that is used to check whether the given input text is matching with the given pattern or not. C# provides a class termed 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* matches 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 fields, but has a constructor that initializes the form, and the following methods:

1. button1\_Click – an event handler that is initiated when a standard 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, 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.

A User has three options after the form is shown:

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

The screenshots of all forms are presented in 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 to get back to it after the registration process was finished, and not to create another instance of Form1. It helps to optimize the program. Storing of Form1 is also needed because Form1 is hidden when the Registration form starts. Form1 can not be closed because if it closes, the whole program shuts down.

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

The 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 user starts to fix problems with invalid data in the fields.
4. textBox2\_TextChanged – an event handler that hides error messages when the user starts to fix problems with invalid data in the fields.
5. textBox3\_TextChanged – an event handler that hides error messages when the user 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

It is used to restore a password if a user forgot it.

It has four fields:

1. From1
2. Users
3. CurrentUsrname
4. SecrA

Form1 field is used for the same purpose, as in the Registration form.

Users field is used to store the list of all users that have already been created.

CurrentUsrname field is used to store the username of a user that wants to restore his/her password.

SecrA field is used to store the answer to the secret question. Then it should be compared to the answer that is stored in the database.

This form has one constructor that sets the value for the Form1 field and initiates an empty list for the Users field.

ForgotPass form has the following methods:

1. button1\_Click – an event handler that resets the password when a user clicks the “Reset” 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. button3\_Click – an event handler that checks if the username exists when a user clicks the “Next” button.

## Form TeacherMainMenu

It is used to restore a password if a user forgot it.

It has two fields:

1. Form1
2. Teacher

Form1 field is used for the same purpose, as in the Registration form.

Teacher field has a type of class Teacher. Therefore, it stores all the information about a teacher that is currently logged in.

This form has one constructor that sets the value for the Form1 field and initiates an empty list for the Users field.

TeacherMainMenu form has the following methods:

1. TeacherMainMenu\_Load – one of the standard event handlers that happens as soon as the form is loaded. In this case, it loads buttons with texts of the groups that have the teacher’s subject this semester.
2. button2\_Click – an event handler that closes the TeacherMainMenu form, and opens Form1 that is stored in the field Form1 when a user clicks the “Go back” button.
3. dynamicBtn\_Click – happens when any of the buttons that were created by TeacherMainMenu\_Load function is clicked. Opens GroupInfo form with the group that is shown in the clicked button.

## Form GroupInfo

It is used to show, create, and edit tests and lectures by the teacher. It also provides the ability for the teacher to get detailed statistics about each of the students in the current group.

It has six fields:

1. CurGr
2. LectDir
3. TestDir
4. TeacherMM
5. rowIndexFromMouseDown
6. rw

CurGr is a field that has a type of class Group. Therefore, it stores the information about the group that the teacher has chosen to in the TeacherMainMenu form. This field provides key information about what lectures, tests, and students to output in dataGridViews.

LectDir is a field that provides a directory to the folder with all the lectures that have been created for the CurGr for this semester.

TestDir is a field that has the same usage as LectDir, but for Tests.

TeacherMM is a field of type TeacherMainMenu that provides a link to the previous form.

rowIndexFromMouseDown and rw fields are used in case if the teacher wants to reorder either lecture or test order.

This form has one constructor that sets the value for the following fields: CurGr, LectDir, TestDir, TeacherMM.

GroupInfo form has the following methods:

1. GroupInfo\_Load – fills the dataGridViews with students, lectures, and tests.
2. button1\_Click\_1 – goes back to the previous form.
3. button2\_Click – opens the Add\_Lecture form.
4. button3\_Click – saves the order of the lectures and tests.
5. button4\_Click – opens the AddTest form.
6. button5\_Click – opens the course project task for editing.
7. dataGridView1\_CellContentClick – opens the StudentStatistics form according to the name of the student in the clicked row.
8. dataGridView2\_CellContentClick – opens the Add\_Lecture form to edit the lecture according to the title of the lecture in the clicked row.
9. dataGridView3\_CellContentClick – opens the AddTest form to edit the test according to the title of the test in the clicked row.
10. dataGridView2\_MouseClick – is used for reordering rows of lectures.
11. dataGridView2\_DragEnter – is used for reordering rows of lectures.
12. dataGridView2\_DragDrop – is used for reordering rows of lectures.
13. dataGridView3\_MouseClick – is used for reordering rows of the test.
14. dataGridView3\_DragEnter – is used for reordering rows of the test.
15. dataGridView3\_DragDrop – is used for reordering rows of the test.

## Form Add\_Lecture

It is used to create and edit lectures.

It has eight fields:

1. CurrentImg
2. CourseProject
3. EditMode
4. Lecture
5. Dir

CurrentImg – current index of the image shown.

CourseProject – boolean that indicates whether this form is used to edit not a lecture, but a course project task.

EditMode – is used to define whether this form is used to create a new lecture, or it should load data of an existing lecture from a file.

GrInfoForm provides a link to the previous form.

Lecture field is used to collect the information about the newly created lecture, or to load the data from an existing lecture.

Dir – a directory without a subfolder of the semester that should contain the lecture.

This form has two constructors: one is used when a new lecture is created, and the other is used when the teacher wants to edit an existing one.

Add\_Lecture form has the following methods:

1. Add\_Lecture\_Load – sets a value for the Dir field, sets filter for the text and image formats that can be imported.
2. button1\_Click – goes back to the previous from.
3. button2\_Click – add image
4. button3\_Click – saves lecture.
5. button4\_Click – import text from a file.
6. button5\_Click – show the previous image.
7. button6\_Click – show the next image.
8. button7\_Click – delete currently shown image.

## Form AddTest

It is used to create and edit lectures.

It has eight fields:

1. EditMode
2. SavedToOrder
3. GrInfoForm
4. Dir
5. OldName
6. Test

EditMode – has the same meaning as in the Add\_Lecture form.

SavedToOrder – a boolean field that defines whether the file has already been added to the order.txt file. Its usage is necessary when a test is being created: the teacher might add some questions and only then save the test, but after going to the Add\_Test\_Question form, the AddTest form is opened with the constructor that assumes that the current test has already been added to the order.txt file. To solve this problem, the SavedToOrder field is used to pass the information across the forms whether the test has or has not been saved.

GrInfoForm – provides a link to the previous form.

Dir – has the same meaning as in the Add\_Lecture form.

OldName – this field is used when a test is being edited. If the title of the test is edited, then the old title must be deleted, and a new one must be written.

Test – has the same meaning as in the Add\_Lecture form, but for the Test class.

This form has two constructors: one is used when a new test is created, and the other is used when the teacher wants to edit an existing one.

AddTest form has the following methods:

1. button1\_Click – goes back to the previous form.
2. button2\_Click – saves test.
3. button3\_Click – opens Add\_Test\_Question form to create a new question.
4. dataGridView1\_CellContentClick – opens the Add\_Test\_Question form to edit a test question according to the title of the test question in the clicked row.

## Form Add\_Test\_Question

It is used to create and edit lectures.

It has four fields:

1. AddTestForm
2. OldQText
3. EditMode

AddTestForm – provides a link to the previous form.

OldQText – has the same meaning as with the OldName field in the AddTest form, but for a test question.

EditMode – has the same meaning as in the Add\_Lecture form.

This form has two constructors: one is used when a new test is created, and the other is used when the teacher wants to edit an existing one.

Add\_Test\_Question form has the following methods:

1. button1\_Click – goes back to the previous form.
2. button2\_Click – saves question.

## Form StudentStatistics

It is used to create and edit lectures.

It has four fields:

1. GroupInfoForm
2. StudentUsrName
3. TotalTime

GroupInfoForm – provides a link to the previous form.

StudentUsrName – the username of the student whose progress is being shown.

TotalTime – the total amount of time that this student has spent on reading lectures, and passing tests of the subject that has the current teacher.

This form has one constructor that sets values to all fields of StudentStatistics class.

StudentStatistics form has the following methods:

1. StudentStatistics\_Load – loads all the marks for and time spent for each lecture and test.
2. button1\_Click – goes back to the previous form.

## Form StudentMainMenu

It is used to restore a password if a user forgot it.

It has two fields:

1. Form1
2. Student

Form1 provides a link to the previous form.

Student field has a type of class Student. Therefore, it stores all the information about a student that is currently logged in.

This form has one constructor that sets values to all fields of StudentMainMenu class.

StudentMainMenu form has the following methods:

1. StudentMainMenu\_Load – loads buttons with texts of the subjects that this student has in this semester.
2. DynamicButton\_Click – happens when any of the buttons that were created by TeacherMainMenu\_Load function is clicked. Opens SubjectTasksStudent form with the group that is shown in the title of the clicked button.
3. button2\_Click – opens the previous form.

## Form SubjectTasksStudent

It is used to restore a password if a user forgot it.

It has four fields:

1. LectDir
2. Subject
3. TestDir
4. StudentMainMenu

LectDir – is a field that provides a directory to the folder with all the lectures that have been created for the group to which the current student belongs.

Subject – stores the information about the subject that the student has chosen in the StudentMainMenu form.

TestDir – is a field that has the same usage as LectDir, but for Tests.

StudentMainMenu – provides a link to the previous form.

This form has one constructor that sets values to all fields of StudentMainMenu class.

SubjectTasksStudent form has the following methods:

1. SubjectTasksStudent\_Load – fills the dataGridViews with lectures and tests.
2. button1\_Click – opens the previous form.
3. button2\_Click – opens the course project task.
4. dataGridView1\_CellContentClick – opens the ViewLecture form to view the lecture according to the title of the lecture in the clicked row.
5. dataGridView2\_CellContentClick – opens the ViewTest form to view the lecture according to the title of the lecture in the clicked row.

## Form ViewLecture

It is used to restore a password if a user forgot it.

It has four fields:

1. CurrentImg
2. StartTime
3. Lecture
4. SubjectTasksStudent

CurrentImg – current index of the image shown.

StartTime – the time when the student opened the lecture.

Lecture – the current lecture that is being presented.

SubjectTasksStudent – provides a link to the previous form.

This form has one constructor that sets values to SubjectTasksStudent and Lecture fields. All the other fields are set automatically.

ViewLecture form has the following methods:

1. ViewLecture\_Load – loads lecture text, title, and pictures to the form.
2. button1\_Click – opens the previous form.
3. button2\_Click – show the previous image.
4. button3\_Click – show the next image.

## Form ViewTest

It is used to restore a password if a user forgot it.

It has four fields:

1. StartTime
2. QuestionInd
3. QuestionMarks
4. NewOrder
5. SubjectTasksStudent
6. Test

StartTime – the time when the student opened the lecture.

QuestionInd – the index of the currently shown question.

QuestionMarks – list of marks that the student receives for each question.

NewOrder – a copy of the original list of questions. It is shuffled if the test requires the order of questions to be randomized.

SubjectTasksStudent – provides a link to the previous form.

Test – the current test that is being presented.

This form has one constructor that sets values to all the fields, except for the StartTime.

ViewTest form has the following methods:

1. ViewTest\_Load – shows the first question and sets the StartTime to the current time.
2. button1\_Click – ends test passing. Exits this form.
3. button6\_Click – show next question.
4. NextQ – functionality that saves the answer to the current question, and shows the next question to the form.
5. saveResult – saves the answer to the current question.

# Structures of Files

All the files with data about users, groups, lectures, and tests can be accessed through the root folder of the project. The root folder of the project is called “Course-project”. All the routes that are going to be presented are going to start with the name of the root folder.

1. File with the list of users can be accessed by the following route:

*Course-project\Course-project\bin\Debug\netcoreapp3.1\users.txt*

1. File with the list of groups that have at least one student in them:

*Course-project\Course-project\bin\Debug\netcoreapp3.1\groupList.txt*

1. The folder with lectures for all specialties can be found with such a route:

*Course-project\Course-project\bin\Debug\netcoreapp3.1\Lectures*

The structure of the Lectures folder look like this:

*Lectures/\*specialty\*/\*year\*/\*sublect\*/\*semester\*/*

Here:

*\*specialty\** – the specialty, for example: 121, 122, 123, etc.

*\*year\** – the year of studying, it can be equal from 1 to 4.

*\*subject\** – the subject of lectures, for example, English, Math, etc.

*\*semester\** – the semester, can be equal from 1 to 2.

Knowing these four variables, a valid directory can be generated to get the needed files of lectures.

The *\*semester\** folder contains files with lectures in *.json* format and the *order.txt* file. This file contains the order in which the lectures should be presented.

1. The Tests folder has an identical folder structure as the Lectures folder.
2. The folder with information about the groups can be found with such a route:

*Course-project\Course-project\bin\Debug\netcoreapp3.1\Groups*

This folder contains folders with names, such as 121.1, 122.4, 172.3, etc. Here, the number before the period indicates the specialty, and the number after – year. Each folder contains a file. Each file has the following structure:

1st line: specialty, for example: 121, 122, 123, etc.

2nd line: year, can be equal from 1 to 4.

3rd line: list of usernames of students that belong to this group separated by a coma.

1. The course project file can be accessed by the following route:

*Course-project\Course-project\bin\Debug\netcoreapp3.1\cp\cp.json*

# 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

### IUser.cs

1. namespace Course\_project {
2. public interface IUser {
3. string Password { get; set; }
4. string SecretAnswer { get; set; }
5. string SecretQuestion { get; set; }
6. string Username { get; set; }
7. }
8. }

### Teacher.cs

1. namespace Course\_project {
2. public class Teacher : IUser {
3. public Teacher(string username, string password, string secretQuestion, string secretAnswer, string subject) {
4. Subject = subject;
5. Username = username;
6. Password = password;
7. SecretQuestion = secretQuestion;
8. SecretAnswer = secretAnswer;
9. }
10. public string Subject { get; set; }
11. public string Password { get; set; }
12. public string SecretAnswer { get; set; }
13. public string SecretQuestion { get; set; }
14. public string Username { get; set; }
15. }
16. }

### Student.cs

1. namespace Course\_project {
2. public class Student : IUser {
3. public Student(string username, string password, string secretQuestion, string secretAnswer, string group) {
4. Group = group;
5. Username = username;
6. Password = password;
7. SecretQuestion = secretQuestion;
8. SecretAnswer = secretAnswer;
9. }
10. public string Group { get; set; }
11. public string Password { get; set; }
12. public string SecretAnswer { get; set; }
13. public string SecretQuestion { get; set; }
14. public string Username { get; set; }
15. }
16. }

### IGroup.cs

1. using System.Collections.Generic;
2. namespace Course\_project {
3. public interface IGroup {
4. string Name { get; set; }
5. string Specialty { get; set; }
6. List<string> Students { get; set; }
7. List<string> Subjects { get; set; }
8. string Year { get; set; }
9. bool addStudent(string username);
10. bool rmStudent(string username);
11. }
12. }

### Group.cs

1. using System;
2. using System.Collections.Generic;
3. using System.IO;
4. namespace Course\_project {
5. public class Group : IGroup {
6. public List<string> Students { get; set; }
7. public List<string> Subjects { get; set; }
8. public string Name { get; set; }
9. public string Specialty { get; set; }
10. public string Year { get; set; }
11. public Group(string grName) {
12. string[] grNameArr = grName.Split(".", StringSplitOptions.RemoveEmptyEntries);
13. Specialty = grNameArr[0];
14. Year = grNameArr[1];
15. Name = grName;
16. Students = new List<string>();
17. Subjects = new List<string>(Rules.getSubjList(Specialty, Year));
18. string grFileDir = "Groups/" + Name + "/" + Name + ".txt";
19. if (File.Exists(@grFileDir)) {
20. StreamReader grDataReader = new StreamReader(File.Open(@grFileDir, FileMode.Open));
21. grDataReader.ReadLine();
22. grDataReader.ReadLine();
23. string[] studenList = grDataReader.ReadLine().Split(",", StringSplitOptions.RemoveEmptyEntries);
24. Students.AddRange(studenList);
25. grDataReader.Close();
26. }
27. }
28. public bool addStudent(string username) {
29. if (!Students.Contains(username)) {
30. Students.Add(username);
31. return true;
32. }
33. return false;
34. }
35. public bool rmStudent(string username) {
36. if (Students.Contains(username)) {
37. Students.Remove(username);
38. return true;
39. }
40. return false;
41. }
42. }
43. }

### ILecture.cs

1. using System.Collections.Generic;
2. namespace Course\_project {
3. public interface ILecture {
4. List<string> ImgList { get; set; }
5. string Name { get; set; }
6. int Semester { get; set; }
7. string Text { get; set; }
8. List<TestMark> StudentMarksList { get; set; }
9. bool addImg(string fileName);
10. void WriteToJson(string dir);
11. }
12. }

### Lecture.cs

1. using Newtonsoft.Json;
2. using System.Collections.Generic;
3. using System.IO;
4. namespace Course\_project {
5. public class Lecture : ILecture {
6. public int Semester { get; set; }
7. public List<string> ImgList { get; set; }
8. public List<TestMark> StudentMarksList { get; set; }
9. public string Name { get; set; }
10. public string Text { get; set; }
11. public Lecture(string name, string text, int semester, List<string> imgList) {
12. Name = name;
13. Text = text;
14. Semester = semester;
15. ImgList = new List<string>(imgList);
16. StudentMarksList = new List<TestMark>();
17. }
18. public Lecture() {
19. Name = null;
20. Text = null;
21. Semester = 1;
22. ImgList = new List<string>();
23. StudentMarksList = new List<TestMark>();
24. }
25. public bool addImg(string fileName) {
26. if (!string.IsNullOrEmpty(fileName)) {
27. ImgList.Add(fileName);
28. return true;
29. }
30. return false;
31. }
32. public void WriteToJson(string dir) {
33. Directory.CreateDirectory(@dir);
34. StreamWriter sw = new StreamWriter(File.Open(@dir + Name + ".json", FileMode.Create));
35. string output = JsonConvert.SerializeObject(this);
36. sw.WriteLine(output);
37. sw.Close();
38. }
39. }
40. }

### ITest.cs

1. using System.Collections.Generic;
2. namespace Course\_project {
3. public interface ITest {
4. bool RandQuestionOrder { get; set; }
5. int Semester { get; set; }
6. List<TestMark> StudentMarksList { get; set; }
7. List<TestQuestion> Questions { get; set; }
8. string Name { get; set; }
9. int getStudentMark(string studentUsrName);
10. int maxScore();
11. List<string> getQuestions();
12. void WriteToJson(string dir);
13. }
14. }

### Test.cs

1. using Newtonsoft.Json;
2. using System.Collections.Generic;
3. using System.IO;
4. namespace Course\_project {
5. public class Test : ITest {
6. public bool RandQuestionOrder { get; set; }
7. public int Semester { get; set; }
8. public List<TestMark> StudentMarksList { get; set; } // List of students who took this test
9. public List<TestQuestion> Questions { get; set; }
10. public string Name { get; set; }
11. public Test(string name, List<TestQuestion> questions, bool randQuestionOrder, int semester) {
12. Name = name;
13. Questions = questions;
14. StudentMarksList = new List<TestMark>();
15. RandQuestionOrder = randQuestionOrder;
16. Semester = semester;
17. }
18. public Test() {
19. Name = null;
20. Questions = new List<TestQuestion>();
21. StudentMarksList = new List<TestMark>();
22. RandQuestionOrder = false;
23. Semester = 1;
24. }
25. public int getStudentMark(string studentUsrName) {
26. foreach (TestMark mark in StudentMarksList)
27. if (string.Compare(mark.StudentUsrName, studentUsrName) == 0)
28. return mark.getStudentMark(maxScore());
29. return 0;
30. }
31. public int maxScore() {
32. int MaxScore = 0;
33. foreach (TestQuestion question in Questions)
34. MaxScore += question.Value;
35. return MaxScore;
36. }
37. public void WriteToJson(string dir) {
38. if (!dir.EndsWith("/"))
39. dir += "/";
40. if (!Directory.Exists(dir))
41. Directory.CreateDirectory(@dir);
42. StreamWriter sw = new StreamWriter(File.Open(@dir + Name + ".json", FileMode.Create));
43. string output = JsonConvert.SerializeObject(this);
44. sw.WriteLine(output);
45. sw.Close();
46. }
47. public List<string> getQuestions() {
48. List<string> qList = new List<string>();
49. foreach (TestQuestion q in Questions)
50. qList.Add(q.Question);
51. return qList;
52. }
53. }
54. }

### ITestQuestion.cs

1. using System.Collections.Generic;
2. namespace Course\_project {
3. public interface ITestQuestion {
4. int Value { get; set; }
5. List<string> RightAns { get; set; }
6. List<string> WrongAns { get; set; }
7. string Question { get; set; }
8. }
9. }

### TestQuestion.cs

1. using System.Collections.Generic;
2. namespace Course\_project {
3. public class TestQuestion : ITestQuestion {
4. public int Value { get; set; } = 1;
5. public string Question { get; set; }
6. public List<string> RightAns { get; set; }
7. public List<string> WrongAns { get; set; }
8. }
9. }

### ITestMark.cs

1. using System;
2. using System.Collections.Generic;
3. namespace Course\_project {
4. public interface ITestMark {
5. List<int> Marks { get; set; }
6. string StudentUsrName { get; set; }
7. TimeSpan TimeSpent { get; set; }
8. int getStudentMark(int maxScore);
9. }
10. }

### TestMark.cs

1. using System;
2. using System.Collections.Generic;
3. namespace Course\_project {
4. public class TestMark : ITestMark {
5. public string StudentUsrName { get; set; }
6. public List<int> Marks { get; set; }
7. public TimeSpan TimeSpent { get; set; }
8. public TestMark() {
9. StudentUsrName = null;
10. Marks = new List<int>();
11. TimeSpent = TimeSpan.Zero;
12. }
13. public TestMark(string studentUsrName, List<int> marks, TimeSpan timeSpent) {
14. StudentUsrName = studentUsrName;
15. Marks = new List<int>(marks);
16. TimeSpent = timeSpent;
17. }
18. public int getStudentMark(int maxScore) {
19. int totalScore = 0;
20. foreach (int mark in Marks)
21. totalScore += mark;
22. totalScore = (totalScore \* 100) / maxScore; // formula to convert to a 100-point scale
23. return totalScore;
24. }
25. }
26. }

### Rules.cs

1. using System;
2. namespace Course\_project {
3. internal class Rules {
4. public static string[,] S121 { get; } = new string[4, 2] {
5. {
6. "English, Math, C++, Git, Systematic View, Physics, History",
7. "English, Math, C#, OOP, Algorithms, Physics, Philosophy"
8. },
9. {
10. "English, Math, C#, OOP, Design, Testing",
11. "English, Math, C#, OOP, Databases, Docker"
12. },
13. {
14. "English, Math, JS, Networks, Databases, Soft Skills",
15. "English, Math, JS, Networks, PM, Soft Skills"
16. },
17. {
18. "English, Math, Python, Project Architecture, Soft Skills",
19. "English, Math, Python, Project Architecture, Soft Skills"
20. }
21. };
22. public static string[,] S122 { get; } = new string[4, 2] {
23. {
24. "English, Math, C++, Git, Systematic View, Physics, History",
25. "English, Math, C#, OOP, Algorithms, Physics, Philosophy"
26. },
27. {
28. "English, Math, C#, OOP, Design, Testing",
29. "English, Math, C#, OOP, Databases, Docker"
30. },
31. {
32. "English, Math, JS, Networks, Databases, Soft Skills",
33. "English, Math, JS, Networks, PM, Soft Skills"
34. },
35. {
36. "English, Math, Python, Project Architecture, Soft Skills",
37. "English, Math, Python, Project Architecture, Soft Skills"
38. }
39. };
40. public static string[,] S123 { get; } = new string[4, 2] {
41. {
42. "English, Math, C++, Git, Systematic View, Physics, History",
43. "English, Math, C#, OOP, Algorithms, Physics, Philosophy"
44. },
45. {
46. "English, Math, C#, OOP, Design, Testing",
47. "English, Math, C#, OOP, Databases, Docker"
48. },
49. {
50. "English, Math, JS, Networks, Databases, Soft Skills",
51. "English, Math, JS, Networks, PM, Soft Skills"
52. },
53. {
54. "English, Math, Python, Project Architecture, Soft Skills",
55. "English, Math, Python, Project Architecture, Soft Skills"
56. }
57. };
58. public static string[,] S172 { get; } = new string[4, 2] {
59. {
60. "English, Math, C++, Git, Systematic View, Physics, History",
61. "English, Math, C#, OOP, Algorithms, Physics, Philosophy"
62. },
63. {
64. "English, Math, C#, OOP, Design, Testing",
65. "English, Math, C#, OOP, Databases, Docker"
66. },
67. {
68. "English, Math, JS, Networks, Databases, Soft Skills",
69. "English, Math, JS, Networks, PM, Soft Skills"
70. },
71. {
72. "English, Math, Python, Project Architecture, Soft Skills",
73. "English, Math, Python, Project Architecture, Soft Skills"
74. }
75. };
76. public static string[] getSubjList(string spec, string year) {
77. int semester = Services.GetCurrentSemester() - 1;
78. int yearConv = Convert.ToInt32(year) - 1;
79. return Convert.ToInt32(spec) switch {
80. 121 => S121[yearConv, semester].Split(","),
81. 122 => S122[yearConv, semester].Split(","),
82. 123 => S123[yearConv, semester].Split(","),
83. 172 => S172[yearConv, semester].Split(","),
84. \_ => null,
85. };
86. }
87. public static string[] getSubjList(string grName) {
88. string[] grNameArr = grName.Split(".", StringSplitOptions.RemoveEmptyEntries);
89. int year = Convert.ToInt32(grNameArr[1]) - 1;
90. int semester = Services.GetCurrentSemester() - 1;
91. return Convert.ToInt32(grNameArr[0]) switch {
92. 121 => S121[year, semester].Split(","),
93. 122 => S122[year, semester].Split(","),
94. 123 => S123[year, semester].Split(","),
95. 172 => S172[year, semester].Split(","),
96. \_ => null,
97. };
98. }
99. }
100. }

### Services.cs

1. using Newtonsoft.Json;
2. using System;
3. using System.Collections.Generic;
4. using System.Drawing;
5. using System.IO;
6. using System.Linq;
7. using System.Windows.Forms;
8. namespace Course\_project {
9. internal class Services {
10. public static int getRand(int start, int end) {
11. Random random = new Random();
12. int i = random.Next(start, end);
13. return i;
14. }
15. public static int GetCurrentSemester() {
16. int semester = DateTime.Now.Month > 6 ? 1 : 2;
17. return semester;
18. }
19. public static List<string> getOrder(string dir) {
20. if (!Directory.Exists(dir))
21. return new List<string>();
22. if (!dir.EndsWith("/"))
23. dir += "/";
24. StreamReader r = new StreamReader(File.Open(@dir + "order.txt", FileMode.OpenOrCreate));
25. List<string> order = new List<string>(r.ReadToEnd().Split(",", StringSplitOptions.RemoveEmptyEntries));
26. r.Close();
27. return order;
28. }
29. public static void rewriteOrder(string dir, List<string> newOrder) {
30. if (!dir.EndsWith("/"))
31. dir += "/";
32. StreamWriter wr = new StreamWriter(File.Open(dir + "order.txt", FileMode.Create));
33. foreach (string lectName in newOrder)
34. wr.Write(lectName + ",");
35. wr.Close();
36. }
37. public static void appendToOrder(string dir, string nameToAppend) {
38. if (!dir.EndsWith("/"))
39. dir += "/";
40. if (!Directory.Exists(dir))
41. Directory.CreateDirectory(dir);
42. StreamWriter wr = new StreamWriter(File.Open(@dir + "order.txt", FileMode.Append));
43. wr.Write(nameToAppend + ",");
44. wr.Close();
45. }
46. public static void saveOrder(string dir, DataGridView dgv) {
47. if (dgv.RowCount - 1 > 0) {
48. StreamWriter wr = new StreamWriter(File.Open(dir + "order.txt", FileMode.Create));
49. for (int i = 0; i < dgv.RowCount - 1; i++)
50. wr.Write((string)dgv.Rows[i].Cells[1].Value + ",");
51. wr.Close();
52. }
53. }
54. public delegate void btnOnClick(object sender, EventArgs e);
55. public static List<Button> createBtnList(int x, int y, List<string> btnTextList, btnOnClick delFunc) {
56. List<Button> btnList = new List<Button>();
57. int posIncrement = 0;
58. Button btn;
59. foreach (string text in btnTextList) {
60. btn = new Button {
61. Location = new Point(x, y + posIncrement),
62. Height = 30,
63. Width = 100,
64. BackColor = Color.White,
65. ForeColor = Color.Black,
66. Text = text,
67. Name = "DynamicButton" + posIncrement,
68. Font = new Font("Georgia", 10)
69. };
70. btn.Click += new EventHandler(delFunc);
71. btnList.Add(btn);
72. posIncrement += 30;
73. }
74. return btnList;
75. }
76. public static List<RadioButton> createRadioBtnList(int x, int y, List<string> btnTextList) {
77. List<RadioButton> radBtnList = new List<RadioButton>();
78. int posIncrement = 0;
79. foreach (string text in btnTextList) {
80. RadioButton radBtn = new RadioButton {
81. Location = new Point(x, y + posIncrement),
82. Text = text,
83. Name = "RadioBtn" + posIncrement,
84. ForeColor = Color.White
85. };
86. radBtnList.Add(radBtn);
87. posIncrement += 20;
88. }
89. return radBtnList;
90. }
91. public static List<CheckBox> createChkBoxList(int x, int y, List<string> btnTextList) {
92. List<CheckBox> radBtnList = new List<CheckBox>();
93. int posIncrement = 0;
94. foreach (string text in btnTextList) {
95. CheckBox chkBox = new CheckBox {
96. Location = new Point(x, y + posIncrement),
97. Text = text,
98. Name = "CheckBox" + posIncrement,
99. ForeColor = Color.White
100. };
101. radBtnList.Add(chkBox);
102. posIncrement += 20;
103. }
104. return radBtnList;
105. }
106. public static List<string> getGroupList() {
107. if (File.Exists(@"groupList.txt")) {
108. StreamReader grListReader = new StreamReader(File.Open(@"groupList.txt", FileMode.Open));
109. List<string> grList = new List<string>(grListReader.ReadToEnd().Split(" ", StringSplitOptions.RemoveEmptyEntries));
110. grListReader.Close();
111. return grList;
112. }
113. return null;
114. }
115. public static List<string> getGroupsWithSubj(string subject) {
116. List<string> grList = getGroupList();
117. if (grList.Count > 0) {
118. List<string> grListSort = new List<string>();
119. foreach (string grName in grList) {
120. List<string> grSubjList = new List<string>(Rules.getSubjList(grName));
121. if (grSubjList.Contains(subject))
122. grListSort.Add(grName);
123. }
124. return grListSort;
125. }
126. return null;
127. }
128. public static T deserializeObj<T>(string filePath) {
129. StreamReader r = new StreamReader(File.Open(@filePath, FileMode.Open));
130. T obj = JsonConvert.DeserializeObject<T>(r.ReadToEnd());
131. r.Close();
132. return obj;
133. }
134. public static void nextImg(ref int currentImg, Lecture lecture, PictureBox pcb) {
135. if (currentImg == -1)
136. return;
137. if (currentImg == lecture.ImgList.Count - 1) {
138. pcb.Image = Image.FromFile(lecture.ImgList[0]);
139. currentImg = 0;
140. } else {
141. pcb.Image = Image.FromFile(lecture.ImgList[currentImg + 1]);
142. currentImg++;
143. }
144. }
145. public static void previousImg(ref int currentImg, Lecture lecture, PictureBox pcb) {
146. if (currentImg == -1)
147. return;
148. if (currentImg == 0) {
149. pcb.Image = Image.FromFile(lecture.ImgList[lecture.ImgList.Count - 1]);
150. currentImg = lecture.ImgList.Count - 1;
151. } else {
152. pcb.Image = Image.FromFile(lecture.ImgList[currentImg - 1]);
153. currentImg--;
154. }
155. }
156. public static List<T> randomizeList<T>(List<T> listToRandomize) {
157. List<T> initialCopy = new List<T>();
158. foreach (T item in listToRandomize)
159. initialCopy.Add(item);
160. List<T> newOrder = new List<T>();
161. int initialListLength = initialCopy.Count;
162. for (int i = 0; i < initialListLength; i++) {
163. int ind = getRand(0, initialCopy.Count);
164. newOrder.Add(initialCopy[ind]);
165. initialCopy.RemoveAt(ind);
166. }
167. return newOrder;
168. }
169. public static void nextQuestion(ViewTest ViewTest, RichTextBox rtb) {
170. TestQuestion currentQ = ViewTest.NewOrder[ViewTest.QuestionInd];
171. if (currentQ.WrongAns.Count == 0) { // detailed answer
172. rtb.Visible = true;
173. } else {
174. List<string> rightWrong = new List<string>(currentQ.RightAns);
175. rightWrong.AddRange(currentQ.WrongAns);
176. List<string> randRightWrong = randomizeList(rightWrong);
177. if (currentQ.RightAns.Count == 1) { // only one right answer
178. List<RadioButton> RadBtnList = createRadioBtnList(50, 130, randRightWrong);
179. foreach (RadioButton rb in RadBtnList)
180. ViewTest.Controls.Add(rb);
181. } else if (currentQ.RightAns.Count > 1) { // multiple right answers
182. List<CheckBox> ChkBoxList = createChkBoxList(50, 130, randRightWrong);
183. foreach (CheckBox cb in ChkBoxList)
184. ViewTest.Controls.Add(cb);
185. }
186. }
187. }
188. public static List<T> getCollection<T>(Form Form) {
189. return new List<T>(Form.Controls.OfType<T>());
190. }
191. public static TestMark derandomizeMarks(ViewTest vt, TimeSpan timeSpent) {
192. TestMark rightMark = new TestMark {
193. Marks = new List<int>(),
194. TimeSpent = timeSpent,
195. StudentUsrName = vt.SubjectTasksStudent.StudentMainMenu.Student.Username
196. };
197. foreach (TestQuestion item in vt.Test.Questions) {
198. int newOrderInd = vt.NewOrder.FindIndex(x => x.Question.Equals(item.Question));
199. rightMark.Marks.Add(vt.QuestionMarks[newOrderInd]);
200. }
201. return rightMark;
202. }
203. public static void openCP(GroupInfo groupInfo) {
204. Add\_Lecture al;
205. if (File.Exists(@"cp/cp.json"))
206. al = new Add\_Lecture(groupInfo, deserializeObj<Lecture>("cp/cp.json"), true);
207. else
208. al = new Add\_Lecture(groupInfo, true);
209. al.Show();
210. groupInfo.Close();
211. }
212. public static string DGVCellContentClick(object sender, DataGridViewCellEventArgs e, int colInd) {
213. DataGridView senderGrid = (DataGridView)sender;
214. if (senderGrid.Columns[e.ColumnIndex] is DataGridViewButtonColumn && e.RowIndex >= 0)
215. return (string)senderGrid.Rows[e.RowIndex].Cells[colInd].Value;
216. return null;
217. }
218. public static void fillDGV(DataGridView dgw, List<string> nameList, string btnText) {
219. int i = 0;
220. foreach (string name in nameList) {
221. dgw.Rows.Add();
222. dgw.Rows[i].Cells[0].Value = i + 1;
223. dgw.Rows[i].Cells[1].Value = name;
224. dgw.Rows[i].Cells[2].Value = btnText;
225. i++;
226. }
227. }
228. // For reordering rows
229. public static void DGVMouseClick(DataGridView dgv, MouseEventArgs e, ref DataGridViewRow Rw, ref int RowIndexFromMouseDown) {
230. if (dgv.SelectedRows.Count == 1)
231. if (e.Button == MouseButtons.Left) {
232. Rw = dgv.SelectedRows[0];
233. RowIndexFromMouseDown = Rw.Index;
234. dgv.DoDragDrop(Rw, DragDropEffects.Move);
235. }
236. }
237. public static void DGVDragEnter(DataGridView dgv, DragEventArgs e) {
238. if (dgv.SelectedRows.Count > 0)
239. e.Effect = DragDropEffects.Move;
240. }
241. public static void DGVragDrop(DataGridView dgv, DragEventArgs e, DataGridViewRow Rw, int RowIndexFromMouseDown) {
242. Point clientPoint = dgv.PointToClient(new Point(e.X, e.Y));
243. if (e.Effect == DragDropEffects.Move) {
244. dgv.Rows.RemoveAt(RowIndexFromMouseDown);
245. dgv.Rows.Insert(dgv.HitTest(clientPoint.X, clientPoint.Y).RowIndex, Rw);
246. }
247. }
248. }
249. }

### RegistrationServices.cs

1. using System.IO;
2. using System.Text.RegularExpressions;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. internal class RegistrationServices {
6. private static readonly Regex regex = new Regex("^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[#?!@$%^&\*-]).{8,}$");
7. public static bool validateUsrName(string usrName) {
8. if (string.IsNullOrEmpty(usrName) || !regex.IsMatch(usrName)) {
9. MessageBox.Show("Username must contain at least 8 symbols, one digit, and one special symbol", "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
10. return false;
11. }
12. BinaryReader reader = new BinaryReader(File.Open(@"users.txt", FileMode.OpenOrCreate));
13. while (reader.BaseStream.Position < reader.BaseStream.Length) { // importing all users
14. string username = reader.ReadString();
15. if (string.Compare(username, usrName) == 0) { // if username taken
16. MessageBox.Show("Username taken!", "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
17. reader.Close();
18. return false;
19. }
20. reader.ReadString();
21. reader.ReadString();
22. reader.ReadString();
23. reader.ReadBoolean();
24. reader.ReadString();
25. }
26. reader.Close();
27. return true;
28. }
29. public static bool validatePass(string password) {
30. if (string.IsNullOrEmpty(password) || !regex.IsMatch(password)) {
31. MessageBox.Show("Password must contain at least 8 symbols, one digit, and one special symbol", "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
32. return false;
33. }
34. return true;
35. }
36. public static bool passMatch(string password, string repeatPass) {
37. if (string.Compare(password, repeatPass) != 0) {
38. MessageBox.Show("Passwords doesn't match!", "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
39. return false;
40. }
41. return true;
42. }
43. }
44. }

### Form1.cs

1. using System;
2. using System.IO;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class Form1 : Form {
6. public Form1() {
7. InitializeComponent();
8. }
9. private void button1\_Click(object sender, EventArgs e) { // Log in
10. string loginUsrname = Convert.ToString(textBox1.Text);
11. string loginPass = Convert.ToString(textBox2.Text);
12. BinaryReader reader = new BinaryReader(File.Open(@"users.txt", FileMode.OpenOrCreate));
13. while (reader.BaseStream.Position < reader.BaseStream.Length) {
14. string usrname = reader.ReadString();
15. string pass = reader.ReadString();
16. string secrQ = reader.ReadString();
17. string secrA = reader.ReadString();
18. bool teacher = reader.ReadBoolean();
19. string subj\_gr = reader.ReadString();
20. if (string.Compare(usrname, loginUsrname) == 0) {
21. if (string.Compare(pass, loginPass) == 0) {
22. if (teacher) {
23. Teacher t = new Teacher(usrname, pass, secrQ, secrA, subj\_gr);
24. TeacherMainMenu tm = new TeacherMainMenu(t, this);
25. tm.Show();
26. textBox1.Clear();
27. textBox2.Clear();
28. reader.Close();
29. Hide();
30. return;
31. } else {
32. Student s = new Student(usrname, pass, secrQ, secrA, subj\_gr);
33. StudentMainMenu sm = new StudentMainMenu(s, this);
34. sm.Show();
35. textBox1.Clear();
36. textBox2.Clear();
37. reader.Close();
38. Hide();
39. return;
40. }
41. } else {
42. MessageBox.Show("Incorrect password!");
43. reader.Close();
44. return;
45. }
46. }
47. }
48. reader.Close();
49. MessageBox.Show("Username not found!");
50. }
51. private void button2\_Click(object sender, EventArgs e) { // Registration
52. Registration r = new Registration(this);
53. r.Show();
54. Hide();
55. }
56. private void button3\_Click(object sender, EventArgs e) { // Forgot password
57. ForgotPass f = new ForgotPass(this);
58. f.Show();
59. Hide();
60. }
61. }
62. }

### Registration.cs

1. using System;
2. using System.Windows.Forms;
3. using System.IO;
4. namespace Course\_project {
5. public partial class Registration : Form {
6. public Form1 Form1 { get; set; }
7. public Registration(Form1 form1) {
8. InitializeComponent();
9. Form1 = form1;
10. }
11. private void button1\_Click(object sender, EventArgs e) {
12. string usrName = Convert.ToString(textBox1.Text);
13. string pass = Convert.ToString(textBox2.Text);
14. string passCheck = Convert.ToString(textBox3.Text);
15. string secrQ = Convert.ToString(comboBox5.Text);
16. string secrA = Convert.ToString(textBox4.Text);
17. string year = Convert.ToString(comboBox1.Text);
18. string spec = Convert.ToString(comboBox2.Text);
19. string subject = Convert.ToString(comboBox3.Text);
20. if (string.IsNullOrEmpty(secrA) || (!radBtn1.Checked && !radBtn2.Checked)) {
21. label4.Visible = true;
22. return;
23. }
24. if (radBtn1.Checked && string.IsNullOrEmpty(subject)) { // if variable fields are empty
25. label4.Visible = true;
26. return;
27. }
28. if (radBtn2.Checked && (string.IsNullOrEmpty(year) || string.IsNullOrEmpty(spec))) { // if variable fields are empty
29. label4.Visible = true;
30. return;
31. }
32. bool usrnameState = RegistrationServices.validateUsrName(usrName);
33. bool passwordState = RegistrationServices.validatePass(pass);
34. bool passMatchState = RegistrationServices.passMatch(pass, passCheck);
35. if (!usrnameState || !passwordState || !passMatchState)
36. return;
37. try {
38. BinaryWriter writer = new BinaryWriter(File.Open(@"users.txt", FileMode.Append));
39. writer.Write(usrName);
40. writer.Write(pass);
41. writer.Write(secrQ);
42. writer.Write(secrA);
43. if (radBtn1.Checked) {
44. writer.Write(true); // true = teacher
45. writer.Write(subject);
46. } else {
47. writer.Write(false); // false = student
48. string grName = spec + "." + year;
49. writer.Write(grName);
50. string dir = "Groups/" + grName + "/";
51. string fName = grName + ".txt";
52. if (Directory.Exists(@dir)) {
53. StreamWriter grWriter = new StreamWriter(File.Open(dir + fName, FileMode.Append));
54. grWriter.Write(", " + usrName);
55. grWriter.Close();
56. } else {
57. Directory.CreateDirectory(dir);
58. StreamWriter grWriter = new StreamWriter(File.Open(dir + fName, FileMode.OpenOrCreate));
59. grWriter.WriteLine(spec);
60. grWriter.WriteLine(year);
61. grWriter.Write(usrName);
62. grWriter.Close();
63. StreamWriter grListWriter = new StreamWriter(File.Open(@"groupList.txt", FileMode.Append));
64. grListWriter.Write(grName + " ");
65. grListWriter.Close();
66. }
67. }
68. writer.Close();
69. } catch (Exception) {
70. throw;
71. }
72. label1.Visible = false;
73. label4.Visible = false;
74. label5.Visible = false;
75. label6.Visible = false;
76. label7.Visible = false;
77. label9.Visible = false;
78. textBox1.Visible = false;
79. textBox2.Visible = false;
80. textBox3.Visible = false;
81. textBox4.Visible = false;
82. comboBox1.Visible = false;
83. comboBox2.Visible = false;
84. comboBox3.Visible = false;
85. comboBox5.Visible = false;
86. button1.Visible = false;
87. panel1.Visible = false;
88. label10.Visible = true;
89. }
90. private void textBox1\_TextChanged(object sender, EventArgs e) {
91. label1.Visible = false;
92. label4.Visible = false;
93. }
94. private void textBox2\_TextChanged(object sender, EventArgs e) {
95. label4.Visible = false;
96. }
97. private void textBox3\_TextChanged(object sender, EventArgs e) {
98. label4.Visible = false;
99. }
100. private void radioButton1\_CheckedChanged(object sender, EventArgs e) {
101. label6.Visible = false;
102. label7.Visible = false;
103. comboBox1.Visible = false;
104. comboBox2.Visible = false;
105. label9.Visible = true;
106. comboBox3.Visible = true;
107. }
108. private void radioButton2\_CheckedChanged(object sender, EventArgs e) {
109. label6.Visible = true;
110. label7.Visible = true;
111. comboBox1.Visible = true;
112. comboBox2.Visible = true;
113. label9.Visible = false;
114. comboBox3.Visible = false;
115. }
116. private void button2\_Click(object sender, EventArgs e) {
117. Form1.Show();
118. Close();
119. }
120. }
121. }

### ForgotPass.cs

1. using System;
2. using System.Collections.Generic;
3. using System.IO;
4. using System.Windows.Forms;
5. namespace Course\_project {
6. public partial class ForgotPass : Form {
7. public Form1 Form1 { get; set; }
8. public List<IUser> Users { get; set; }
9. public string CurrentUsrname { get; set; }
10. public string SecrA { get; set; }
11. public ForgotPass(Form1 form1) {
12. InitializeComponent();
13. Users = new List<IUser>();
14. Form1 = form1;
15. }
16. private void button1\_Click(object sender, EventArgs e) { // reset button
17. if (string.Compare(Convert.ToString(textBox2.Text), SecrA) != 0) { // secret answer comparison
18. label4.Visible = true;
19. return;
20. }
21. string newPass = Convert.ToString(textBox3.Text);
22. bool passwordState = RegistrationServices.validatePass(newPass);
23. if (!passwordState) {
24. label5.Visible = true;
25. return;
26. }
27. foreach (IUser user in Users)
28. if (string.Compare(user.Username, CurrentUsrname) == 0)
29. user.Password = newPass;
30. // Change the binary file
31. foreach (IUser user in Users)
32. try {
33. BinaryWriter writer = new BinaryWriter(File.Open(@"users.txt", FileMode.Truncate));
34. writer.Write(user.Username);
35. writer.Write(user.Password);
36. writer.Write(user.SecretQuestion);
37. writer.Write(user.SecretAnswer);
38. if (user is Teacher) {
39. writer.Write(true); // true = teacher
40. writer.Write((user as Teacher).Subject);
41. } else {
42. writer.Write(false); // false = student
43. writer.Write((user as Student).Group);
44. }
45. writer.Close();
46. } catch (Exception) {
47. throw;
48. }
49. label1.Visible = false;
50. label3.Visible = false;
51. button1.Visible = false;
52. textBox1.Visible = false;
53. textBox2.Visible = false;
54. textBox3.Visible = false;
55. label2.Visible = true;
56. }
57. private void button2\_Click(object sender, EventArgs e) { // Go back button
58. Form1.Show();
59. Close();
60. }
61. private void button3\_Click(object sender, EventArgs e) { // next button
62. CurrentUsrname = Convert.ToString(textBox1.Text);
63. BinaryReader reader = new BinaryReader(File.Open(@"users.txt", FileMode.OpenOrCreate));
64. bool found = false;
65. string secrQ = "";
66. while (reader.BaseStream.Position < reader.BaseStream.Length) { // importing all users
67. string usrname = reader.ReadString();
68. string pass = reader.ReadString();
69. secrQ = reader.ReadString();
70. SecrA = reader.ReadString();
71. bool teacher = reader.ReadBoolean();
72. string subj\_gr = reader.ReadString();
73. IUser u;
74. if (teacher)
75. u = new Teacher(usrname, pass, secrQ, SecrA, subj\_gr);
76. else
77. u = new Student(usrname, pass, secrQ, SecrA, subj\_gr);
78. Users.Add(u);
79. if (string.Compare(usrname, CurrentUsrname) == 0) // if username found
80. found = true;
81. }
82. reader.Close();
83. if (found) {
84. textBox1.Enabled = false;
85. button3.Visible = false;
86. textBox2.Visible = true;
87. textBox3.Visible = true;
88. button1.Visible = true;
89. label3.Text = secrQ;
90. label3.Visible = true;
91. return;
92. } else {
93. MessageBox.Show("No such username!");
94. return;
95. }
96. }
97. }
98. }

### TeacherMainMenu.cs

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class TeacherMainMenu : Form {
6. public Form1 Form1 { get; set; }
7. public Teacher Teacher { get; set; }
8. public TeacherMainMenu(Teacher teacher, Form1 form1) {
9. InitializeComponent();
10. Teacher = teacher;
11. Form1 = form1;
12. }
13. private void TeacherMainMenu\_Load(object sender, EventArgs e) {
14. label1.Text = Teacher.Subject;
15. label2.Text = "Hello, " + Teacher.Username;
16. List<string> grListSort = Services.getGroupsWithSubj(Teacher.Subject);
17. List<Button> btnList = Services.createBtnList(20, 150, grListSort, dynamicBtn\_Click);
18. foreach (Button btn in btnList)
19. Controls.Add(btn);
20. }
21. private void dynamicBtn\_Click(object sender, EventArgs e) {
22. string grName = (sender as Button).Text; // get text of the btn that was clicked
23. GroupInfo grInfo = new GroupInfo(grName, this);
24. grInfo.Show();
25. Hide();
26. }
27. private void button2\_Click(object sender, EventArgs e) {
28. Form1.Show();
29. Close();
30. }
31. }
32. }

### GroupInfo.cs

1. using System;
2. using System.Windows.Forms;
3. namespace Course\_project {
4. public partial class GroupInfo : Form {
5. public Group CurGr { get; set; }
6. public string LectDir { get; set; } // with semester
7. public string TestDir { get; set; } // with semester
8. public TeacherMainMenu TeacherMM { get; set; }
9. private int rowIndexFromMouseDown; // For reordering rows
10. private DataGridViewRow rw;
11. public GroupInfo(string grName, TeacherMainMenu teacherMM) {
12. InitializeComponent();
13. CurGr = new Group(grName);
14. TeacherMM = teacherMM;
15. int semester = Services.GetCurrentSemester();
16. LectDir = "Lectures/" + CurGr.Specialty + "/" + CurGr.Year + "/" + teacherMM.Teacher.Subject + "/" + semester + "/";
17. TestDir = "Tests/" + CurGr.Specialty + "/" + CurGr.Year + "/" + teacherMM.Teacher.Subject + "/" + semester + "/";
18. }
19. private void GroupInfo\_Load(object sender, EventArgs e) {
20. label1.Text = CurGr.Name;
21. Services.fillDGV(dataGridView1, CurGr.Students, "Statistics");
22. Services.fillDGV(dataGridView2, Services.getOrder(LectDir), "Edit");
23. Services.fillDGV(dataGridView3, Services.getOrder(TestDir), "Edit");
24. if (string.Compare(TeacherMM.Teacher.Subject, "English") == 0 && string.Compare(CurGr.Name, "121.2") == 0)
25. button5.Visible = true;
26. }
27. private void button1\_Click\_1(object sender, EventArgs e) { // Back
28. TeacherMM.Show();
29. Close();
30. }
31. private void button2\_Click(object sender, EventArgs e) { // Add lecture
32. Add\_Lecture al = new Add\_Lecture(this);
33. al.Show();
34. Close();
35. }
36. private void button3\_Click(object sender, EventArgs e) { // Save order
37. Services.saveOrder(LectDir, dataGridView2);
38. Services.saveOrder(TestDir, dataGridView3);
39. MessageBox.Show("New order succesfully saved!", "", MessageBoxButtons.OK, MessageBoxIcon.Information);
40. }
41. private void button4\_Click(object sender, EventArgs e) { // Add test
42. AddTest at = new AddTest(this);
43. at.Show();
44. Close();
45. }
46. private void button5\_Click(object sender, EventArgs e) { // Course project
47. Services.openCP(this);
48. }
49. private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e) {
50. string studentUsrName = Services.DGVCellContentClick(sender, e, 1);
51. if (!string.IsNullOrEmpty(studentUsrName)) {
52. StudentStatistics ss = new StudentStatistics(this, studentUsrName);
53. ss.Show();
54. Hide();
55. }
56. }
57. private void dataGridView2\_CellContentClick(object sender, DataGridViewCellEventArgs e) { // "Edit" btn OnClick in lecture list
58. string lectName = Services.DGVCellContentClick(sender, e, 1);
59. if (!string.IsNullOrEmpty(lectName)) {
60. Lecture lect = Services.deserializeObj<Lecture>(LectDir + lectName + ".json");
61. Add\_Lecture al = new Add\_Lecture(this, lect);
62. al.Show();
63. Close();
64. }
65. }
66. private void dataGridView3\_CellContentClick(object sender, DataGridViewCellEventArgs e) { // "Edit" btn OnClick in test list
67. string testName = Services.DGVCellContentClick(sender, e, 1);
68. if (!string.IsNullOrEmpty(testName)) {
69. Test test = Services.deserializeObj<Test>(TestDir + testName + ".json");
70. AddTest at = new AddTest(this, test, true);
71. at.Show();
72. Close();
73. }
74. }
75. // For reordering rows of lectures
76. private void dataGridView2\_MouseClick(object sender, MouseEventArgs e) {
77. Services.DGVMouseClick((DataGridView)sender, e, ref rw, ref rowIndexFromMouseDown);
78. }
79. private void dataGridView2\_DragEnter(object sender, DragEventArgs e) {
80. Services.DGVDragEnter((DataGridView)sender, e);
81. }
82. private void dataGridView2\_DragDrop(object sender, DragEventArgs e) {
83. Services.DGVragDrop((DataGridView)sender, e, rw, rowIndexFromMouseDown);
84. }
85. // For reordering rows of tests
86. private void dataGridView3\_MouseClick(object sender, MouseEventArgs e) {
87. Services.DGVMouseClick((DataGridView)sender, e, ref rw, ref rowIndexFromMouseDown);
88. }
89. private void dataGridView3\_DragEnter(object sender, DragEventArgs e) {
90. Services.DGVDragEnter((DataGridView)sender, e);
91. }
92. private void dataGridView3\_DragDrop(object sender, DragEventArgs e) {
93. Services.DGVragDrop((DataGridView)sender, e, rw, rowIndexFromMouseDown);
94. }
95. }
96. }

### Add\_Lecture.cs

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.IO;
5. using System.Windows.Forms;
6. namespace Course\_project {
7. public partial class Add\_Lecture : Form {
8. private int currentImg;
9. public bool CourseProject { get; set; }
10. public bool EditMode { get; set; }
11. public GroupInfo GrInfoForm { get; set; }
12. public int CurrentImg { get => currentImg; set => currentImg = value; }
13. public Lecture Lecture { get; set; }
14. public string Dir { get; set; } // without semester
15. public Add\_Lecture(GroupInfo grInfoForm, bool courseProject = false) { // Constructor for creating a new lect
16. InitializeComponent();
17. GrInfoForm = grInfoForm;
18. EditMode = false;
19. CurrentImg = -1;
20. Lecture = new Lecture();
21. CourseProject = courseProject;
22. comboBox3.SelectedIndex = 0;
23. }
24. public Add\_Lecture(GroupInfo grInfoForm, Lecture lecture, bool courseProject = false) { // Constructor for editing a lect
25. InitializeComponent();
26. Text = "Edit lecture";
27. GrInfoForm = grInfoForm;
28. Lecture = lecture;
29. EditMode = true;
30. CourseProject = courseProject;
31. if (lecture.ImgList.Count != 0) { // check if empty
32. CurrentImg = 0;
33. pictureBox1.Image = Image.FromFile(lecture.ImgList[CurrentImg]);
34. } else {
35. CurrentImg = -1;
36. lecture.ImgList = new List<string>();
37. }
38. comboBox3.Text = Convert.ToString(lecture.Semester);
39. textBox1.Text = lecture.Name;
40. richTextBox1.Text = lecture.Text;
41. }
42. private void Add\_Lecture\_Load(object sender, EventArgs e) {
43. Dir = "Lectures/" + GrInfoForm.CurGr.Specialty + "/" + GrInfoForm.CurGr.Year + "/" + GrInfoForm.TeacherMM.Teacher.Subject + "/";
44. openFileDialog1.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";
45. openFileDialog2.Filter = "Image Files(\*.BMP; \*.JPG; \*.GIF)| \*.BMP; \*.JPG; \*.GIF | All files(\*.\*) | \*.\*";
46. if (CourseProject) {
47. label1.Visible = false;
48. label2.Visible = false;
49. textBox1.Visible = false;
50. comboBox3.Visible = false;
51. }
52. }
53. private void button1\_Click(object sender, EventArgs e) { // Back
54. GroupInfo grInfo = new GroupInfo(GrInfoForm.CurGr.Name, GrInfoForm.TeacherMM);
55. grInfo.Show();
56. Close();
57. }
58. private void button2\_Click(object sender, EventArgs e) { // Add img
59. if (openFileDialog2.ShowDialog() == DialogResult.Cancel)
60. return;
61. string imgName = openFileDialog2.FileName;
62. Lecture.ImgList.Add(imgName);
63. pictureBox1.Image = Image.FromFile(imgName);
64. if (CurrentImg == -1)
65. CurrentImg++;
66. else
67. CurrentImg = Lecture.ImgList.Count - 1;
68. }
69. private void button3\_Click(object sender, EventArgs e) { // Save
70. string name = Convert.ToString(textBox1.Text);
71. string newText = Convert.ToString(richTextBox1.Text);
72. int newSemester = Convert.ToInt32(comboBox3.SelectedItem);
73. if (CourseProject) {
74. Lecture cp = new Lecture("cp", newText, 0, Lecture.ImgList);
75. cp.WriteToJson("cp/");
76. MessageBox.Show("Course project succesfuly saved");
77. GroupInfo grInfoForm = new GroupInfo(GrInfoForm.CurGr.Name, GrInfoForm.TeacherMM);
78. grInfoForm.Show();
79. Close();
80. return;
81. }
82. if (string.IsNullOrEmpty(name) || string.IsNullOrEmpty(newText)) {
83. MessageBox.Show("Fill in all cells!");
84. return;
85. }
86. if (EditMode) {
87. List<string> lectOrder = Services.getOrder(Dir + Lecture.Semester);
88. int lectInd = lectOrder.FindIndex(x => x.Equals(Lecture.Name));
89. File.Delete(Dir + Lecture.Semester + "/" + lectOrder[lectInd] + ".json"); // delete old version
90. if (newSemester == Lecture.Semester) {
91. lectOrder[lectInd] = name;
92. Services.rewriteOrder(Dir + newSemester, lectOrder);
93. } else {
94. lectOrder.RemoveAt(lectInd); // delete the lecture because it's moved to another semester
95. Services.rewriteOrder(Dir + Lecture.Semester, lectOrder);
96. Services.appendToOrder(Dir + newSemester, name);
97. }
98. } else
99. Services.appendToOrder(Dir + newSemester, name);
100. Lecture lect = new Lecture(name, newText, newSemester, Lecture.ImgList);
101. lect.WriteToJson(Dir + newSemester + "/");
102. MessageBox.Show("The lecture is succesfuly saved");
103. GroupInfo grInfo = new GroupInfo(GrInfoForm.CurGr.Name, GrInfoForm.TeacherMM);
104. grInfo.Show();
105. Close();
106. }
107. private void button4\_Click(object sender, EventArgs e) { // Import text
108. if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
109. return;
110. string filename = openFileDialog1.FileName;
111. string fileText = File.ReadAllText(filename);
112. richTextBox1.Text = fileText;
113. }
114. private void button5\_Click(object sender, EventArgs e) {
115. Services.previousImg(ref currentImg, Lecture, pictureBox1);
116. }
117. private void button6\_Click(object sender, EventArgs e) {
118. Services.nextImg(ref currentImg, Lecture, pictureBox1);
119. }
120. private void button7\_Click(object sender, EventArgs e) { // Delete img
121. if (CurrentImg == -1)
122. return;
123. if (Lecture.ImgList.Count == 1) {
124. Lecture.ImgList.RemoveAt(CurrentImg);
125. pictureBox1.Image = null;
126. CurrentImg = -1;
127. return;
128. }
129. if (CurrentImg == Lecture.ImgList.Count - 1) {
130. Lecture.ImgList.RemoveAt(CurrentImg);
131. pictureBox1.Image = Image.FromFile(Lecture.ImgList[0]);
132. CurrentImg = 0;
133. return;
134. } else if (CurrentImg == 0) {
135. Lecture.ImgList.RemoveAt(CurrentImg);
136. pictureBox1.Image = Image.FromFile(Lecture.ImgList[Lecture.ImgList.Count - 1]);
137. CurrentImg = Lecture.ImgList.Count - 1;
138. return;
139. } else {
140. Lecture.ImgList.RemoveAt(CurrentImg);
141. pictureBox1.Image = Image.FromFile(Lecture.ImgList[CurrentImg + 1]);
142. CurrentImg++;
143. return;
144. }
145. }
146. }
147. }

### AddTest.cs

1. using System;
2. using System.Collections.Generic;
3. using System.IO;
4. using System.Windows.Forms;
5. namespace Course\_project {
6. public partial class AddTest : Form {
7. public bool EditMode { get; set; }
8. public bool SavedToOrder { get; set; } // true when this from is loaded for the first
9. public GroupInfo GrInfoForm { get; set; }
10. public string Dir { get; set; } // without semester
11. public string OldName { get; set; }
12. public Test Test { get; set; }
13. public AddTest(GroupInfo grInfoForm) { // Constructor for creating new tests
14. InitializeComponent();
15. GrInfoForm = grInfoForm;
16. Test = new Test();
17. EditMode = false;
18. SavedToOrder = false;
19. Dir = "Tests/" + grInfoForm.CurGr.Specialty + "/" + grInfoForm.CurGr.Year + "/" + grInfoForm.TeacherMM.Teacher.Subject + "/";
20. comboBox3.SelectedIndex = 0;
21. }
22. public AddTest(GroupInfo grInfoForm, Test test, bool savedToOrder = false) { // Constructor for editing tests
23. InitializeComponent();
24. GrInfoForm = grInfoForm;
25. Test = test;
26. SavedToOrder = savedToOrder;
27. EditMode = true;
28. OldName = test.Name;
29. Dir = "Tests/" + grInfoForm.CurGr.Specialty + "/" + grInfoForm.CurGr.Year + "/" + grInfoForm.TeacherMM.Teacher.Subject + "/";
30. textBox1.Text = test.Name;
31. comboBox3.Text = Convert.ToString(test.Semester);
32. checkBox1.Checked = test.RandQuestionOrder == true ? true : false;
33. if (savedToOrder)
34. Text = "Edit test";
35. Services.fillDGV(dataGridView1, test.getQuestions(), "View");
36. }
37. private void button1\_Click(object sender, EventArgs e) { // Back
38. GroupInfo grInfo = new GroupInfo(GrInfoForm.CurGr.Name, GrInfoForm.TeacherMM);
39. grInfo.Show();
40. Close();
41. }
42. private void button2\_Click(object sender, EventArgs e) { // Save
43. string name = textBox1.Text;
44. bool randQOrder = checkBox1.Checked == true ? true : false;
45. int newSemester = Convert.ToInt32(comboBox3.SelectedItem);
46. if (string.IsNullOrEmpty(name)) {
47. MessageBox.Show("Fill in test name field!");
48. return;
49. }
50. if (EditMode && SavedToOrder) {
51. List<string> testOrder = Services.getOrder(Dir + Test.Semester);
52. int testInd = testOrder.FindIndex(x => x.Equals(OldName));
53. File.Delete(Dir + Test.Semester + "/" + testOrder[testInd] + ".json"); // delete old version
54. if (newSemester == Test.Semester) {
55. testOrder[testInd] = name;
56. Services.rewriteOrder(Dir + newSemester, testOrder);
57. } else {
58. testOrder.RemoveAt(testInd); // delete the lecture because it's moved to another semester
59. Services.rewriteOrder(Dir + Test.Semester, testOrder);
60. Services.appendToOrder(Dir + newSemester, name);
61. }
62. } else
63. Services.appendToOrder(Dir + newSemester, name);
64. Test newTest = new Test(name, Test.Questions, randQOrder, newSemester);
65. newTest.WriteToJson(Dir + newSemester);
66. MessageBox.Show("The test is succesfuly saved!");
67. GroupInfo grInfo = new GroupInfo(GrInfoForm.CurGr.Name, GrInfoForm.TeacherMM);
68. grInfo.Show();
69. Close();
70. }
71. private void button3\_Click(object sender, EventArgs e) { // Add question
72. Add\_Test\_Question atq = new Add\_Test\_Question(this);
73. atq.Show();
74. Hide();
75. }
76. private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e) { // "Edit" btn OnClick in question list
77. DataGridView senderGrid = (DataGridView)sender;
78. if (senderGrid.Columns[e.ColumnIndex] is DataGridViewButtonColumn && e.RowIndex >= 0) {
79. string clickedQText = (string)senderGrid.Rows[e.RowIndex].Cells[1].Value;
80. TestQuestion clickedQ = Test.Questions.Find(x => x.Question.Contains(clickedQText));
81. Add\_Test\_Question tq = new Add\_Test\_Question(this, clickedQ);
82. tq.Show();
83. Hide();
84. }
85. }
86. }
87. }

### Add\_Test\_Question

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class Add\_Test\_Question : Form {
6. public AddTest AddTestForm { get; set; }
7. public string OldQText { get; set; }
8. public bool EditMode { get; set; }
9. public Add\_Test\_Question(AddTest addTestForm) { // Constructor for creating new questions
10. InitializeComponent();
11. AddTestForm = addTestForm;
12. EditMode = false;
13. comboBox1.SelectedIndex = 0;
14. }
15. public Add\_Test\_Question(AddTest addTestForm, TestQuestion question) { // Constructor for editing questions
16. InitializeComponent();
17. Text = "Edit question";
18. AddTestForm = addTestForm;
19. EditMode = true;
20. OldQText = question.Question;
21. textBox1.Text = question.Question;
22. comboBox1.SelectedIndex = question.Value - 1;
23. foreach (string ra in question.RightAns)
24. richTextBox1.Text += ra + "\n";
25. foreach (string wa in question.WrongAns)
26. richTextBox2.Text += wa + "\n";
27. }
28. private void button1\_Click(object sender, EventArgs e) { // Back
29. AddTest at = new AddTest(AddTestForm.GrInfoForm, AddTestForm.Test, AddTestForm.SavedToOrder);
30. at.Show();
31. Close();
32. }
33. private void button2\_Click(object sender, EventArgs e) { // Save
34. if (string.IsNullOrEmpty(textBox1.Text) || (string.IsNullOrEmpty(richTextBox1.Text) && string.IsNullOrEmpty(richTextBox1.Text))) {
35. MessageBox.Show("Fill in all cells!");
36. return;
37. }
38. TestQuestion tq = new TestQuestion {
39. Question = textBox1.Text,
40. Value = Convert.ToInt32(comboBox1.SelectedItem),
41. RightAns = new List<string>(Convert.ToString(richTextBox1.Text).Split("\n", StringSplitOptions.RemoveEmptyEntries)),
42. WrongAns = new List<string>(Convert.ToString(richTextBox2.Text).Split("\n", StringSplitOptions.RemoveEmptyEntries))
43. };
44. if (EditMode) {
45. for (int i = 0; i < AddTestForm.Test.Questions.Count; i++)
46. if (string.Compare(OldQText, AddTestForm.Test.Questions[i].Question) == 0) {
47. AddTestForm.Test.Questions[i] = tq;
48. break;
49. }
50. } else
51. AddTestForm.Test.Questions.Add(tq);
52. MessageBox.Show("Question is succesfuly saved!");
53. AddTest at = new AddTest(AddTestForm.GrInfoForm, AddTestForm.Test, AddTestForm.SavedToOrder);
54. at.Show();
55. Close();
56. }
57. }
58. }

### StudentStatistics.cs

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class StudentStatistics : Form {
6. public GroupInfo GroupInfoForm { get; set; }
7. public string StudentUsrName { get; set; }
8. public TimeSpan TotalTime { get; set; }
9. public StudentStatistics(GroupInfo groupInfoForm, string studentUsrName) {
10. InitializeComponent();
11. GroupInfoForm = groupInfoForm;
12. StudentUsrName = studentUsrName;
13. TotalTime = TimeSpan.Zero;
14. }
15. private void StudentStatistics\_Load(object sender, EventArgs e) {
16. List<string> lectOrder = Services.getOrder(GroupInfoForm.LectDir);
17. List<string> testOrder = Services.getOrder(GroupInfoForm.TestDir);
18. Services.fillDGV(dataGridView1, lectOrder, "");
19. Services.fillDGV(dataGridView2, testOrder, "");
20. int i = 0;
21. foreach (string lectName in lectOrder) {
22. Lecture l = Services.deserializeObj<Lecture>(GroupInfoForm.LectDir + lectName + ".json");
23. int studentIndex = l.StudentMarksList.FindIndex(x => x.StudentUsrName.Equals(StudentUsrName));
24. if (studentIndex != -1) {
25. TimeSpan ts = l.StudentMarksList[studentIndex].TimeSpent;
26. dataGridView1.Rows[i].Cells[2].Value = ts.Hours + ":" + ts.Minutes + ":" + ts.Seconds;
27. TotalTime += l.StudentMarksList[studentIndex].TimeSpent;
28. } else
29. dataGridView1.Rows[i].Cells[2].Value = "Not read";
30. i++;
31. }
32. i = 0;
33. foreach (string testName in testOrder) {
34. Test t = Services.deserializeObj<Test>(GroupInfoForm.TestDir + testName + ".json");
35. int studentIndex = t.StudentMarksList.FindIndex(x => x.StudentUsrName.Equals(StudentUsrName));
36. if (studentIndex != -1) {
37. TimeSpan ts = t.StudentMarksList[studentIndex].TimeSpent;
38. dataGridView2.Rows[i].Cells[2].Value = ts.Hours + ":" + ts.Minutes + ":" + ts.Seconds;
39. dataGridView2.Rows[i].Cells[3].Value = t.getStudentMark(StudentUsrName);
40. TotalTime += t.StudentMarksList[studentIndex].TimeSpent;
41. } else {
42. dataGridView2.Rows[i].Cells[2].Value = "Not passed";
43. dataGridView2.Rows[i].Cells[3].Value = "Not passed";
44. }
45. i++;
46. }
47. label1.Text += TotalTime.Hours + ":" + TotalTime.Minutes + ":" + TotalTime.Seconds;
48. }
49. private void button1\_Click(object sender, EventArgs e) {
50. GroupInfoForm.Show();
51. Close();
52. }
53. }
54. }

### StudentMainMenu.cs

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class StudentMainMenu : Form {
6. public Form1 Form1 { get; set; }
7. public Student Student { get; set; }
8. public StudentMainMenu(Student student, Form1 form1) {
9. InitializeComponent();
10. Form1 = form1;
11. Student = student;
12. }
13. private void StudentMainMenu\_Load(object sender, EventArgs e) {
14. label1.Text = Student.Group;
15. label2.Text = "Hello, " + Student.Username;
16. List<string> grSubjList = new List<string>(Rules.getSubjList(Student.Group));
17. List<Button> btnList = Services.createBtnList(80, 150, grSubjList, DynamicButton\_Click);
18. foreach (Button btn in btnList)
19. Controls.Add(btn);
20. }
21. private void DynamicButton\_Click(object sender, EventArgs e) {
22. string subject = (sender as Button).Text;
23. SubjectTasksStudent STS = new SubjectTasksStudent(subject, this);
24. STS.Show();
25. Hide();
26. }
27. private void button2\_Click(object sender, EventArgs e) {
28. Form1.Show();
29. Close();
30. }
31. }
32. }

### SubjectTasksStudent.cs

1. using System;
2. using System.Windows.Forms;
3. namespace Course\_project {
4. public partial class SubjectTasksStudent : Form {
5. public string LectDir { get; set; } // with semester
6. public string Subject { get; set; }
7. public string TestDir { get; set; } // with semester
8. public StudentMainMenu StudentMainMenu { get; set; }
9. public SubjectTasksStudent(string subject, StudentMainMenu studentMainMenu) {
10. InitializeComponent();
11. StudentMainMenu = studentMainMenu;
12. Subject = subject;
13. Group group = new Group(StudentMainMenu.Student.Group);
14. int semester = Services.GetCurrentSemester();
15. LectDir = "Lectures/" + group.Specialty + "/" + group.Year + "/" + Subject + "/" + semester + "/";
16. TestDir = "Tests/" + group.Specialty + "/" + group.Year + "/" + Subject + "/" + semester + "/";
17. }
18. private void SubjectTasksStudent\_Load(object sender, EventArgs e) {
19. label1.Text = Subject;
20. Services.fillDGV(dataGridView1, Services.getOrder(LectDir), "View");
21. Services.fillDGV(dataGridView2, Services.getOrder(TestDir), "Start");
22. if (string.Compare(Subject, "English") == 0 && string.Compare(StudentMainMenu.Student.Group, "121.2") == 0 && DateTime.Now.Month >= 9 && DateTime.Now.Month < 12)
23. if (!(DateTime.Now.Month == 9 && DateTime.Now.Day < 21))
24. button2.Visible = true;
25. }
26. private void button1\_Click(object sender, EventArgs e) {
27. StudentMainMenu.Show();
28. Close();
29. }
30. private void button2\_Click(object sender, EventArgs e) {
31. Lecture cp = Services.deserializeObj<Lecture>("cp/cp.json");
32. ViewLecture vl = new ViewLecture(this, cp);
33. vl.Show();
34. Hide();
35. }
36. private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e) {
37. string lectName = Services.DGVCellContentClick(sender, e, 1);
38. if (!string.IsNullOrEmpty(lectName)) {
39. Lecture lect = Services.deserializeObj<Lecture>(LectDir + lectName + ".json");
40. ViewLecture vl = new ViewLecture(this, lect);
41. vl.Show();
42. Hide();
43. }
44. }
45. private void dataGridView2\_CellContentClick(object sender, DataGridViewCellEventArgs e) {
46. string testName = Services.DGVCellContentClick(sender, e, 1);
47. if (!string.IsNullOrEmpty(testName)) {
48. Test test = Services.deserializeObj<Test>(TestDir + testName + ".json");
49. if (test.StudentMarksList.Exists(x => x.StudentUsrName == StudentMainMenu.Student.Username)) {
50. MessageBox.Show("You have already passed this test");
51. return;
52. }
53. if (test.Questions.Count == 0) {
54. MessageBox.Show("Test isn't ready yet!");
55. return;
56. }
57. ViewTest vt = new ViewTest(this, test);
58. vt.Show();
59. Hide();
60. }
61. }
62. }
63. }

### ViewLecture.cs

1. using System;
2. using System.Drawing;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class ViewLecture : Form {
6. private int currentImg;
7. public int CurrentImg { get => currentImg; set => currentImg = value; }
8. public DateTime StartTime { get; set; }
9. public Lecture Lecture { get; set; }
10. public SubjectTasksStudent SubjectTasksStudent { get; set; }
11. public ViewLecture(SubjectTasksStudent subjectTasksStudent, Lecture lecture) {
12. InitializeComponent();
13. SubjectTasksStudent = subjectTasksStudent;
14. Lecture = lecture;
15. }
16. private void ViewLecture\_Load(object sender, EventArgs e) {
17. label1.Text = Lecture.Name;
18. richTextBox1.Text = Lecture.Text;
19. if (Lecture.ImgList.Count != 0) {
20. CurrentImg = 0;
21. pictureBox1.Image = Image.FromFile(Lecture.ImgList[CurrentImg]);
22. } else
23. CurrentImg = -1;
24. StartTime = DateTime.Now;
25. }
26. private void button1\_Click(object sender, EventArgs e) {
27. TimeSpan timeSpent = DateTime.Now - StartTime;
28. TestMark tm;
29. int studentIndex = Lecture.StudentMarksList.FindIndex(x => x.StudentUsrName.Equals(SubjectTasksStudent.StudentMainMenu.Student.Username));
30. if (studentIndex != -1)
31. Lecture.StudentMarksList[studentIndex].TimeSpent += timeSpent;
32. else {
33. tm = new TestMark {
34. StudentUsrName = SubjectTasksStudent.StudentMainMenu.Student.Username,
35. TimeSpent = timeSpent
36. };
37. Lecture.StudentMarksList.Add(tm);
38. }
39. Lecture.WriteToJson(SubjectTasksStudent.LectDir);
40. SubjectTasksStudent.Show();
41. Close();
42. }
43. private void button2\_Click(object sender, EventArgs e) {
44. Services.previousImg(ref currentImg, Lecture, pictureBox1);
45. }
46. private void button3\_Click(object sender, EventArgs e) {
47. Services.nextImg(ref currentImg, Lecture, pictureBox1);
48. }
49. }
50. }

### ViewTest.cs

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. namespace Course\_project {
5. public partial class ViewTest : Form {
6. public DateTime StartTime { get; set; }
7. public int QuestionInd { get; set; }
8. public List<int> QuestionMarks { get; set; }
9. public List<TestQuestion> NewOrder { get; set; }
10. public SubjectTasksStudent SubjectTasksStudent { get; set; }
11. public Test Test { get; set; }
12. public ViewTest(SubjectTasksStudent subjectTasksStudent, Test test) {
13. InitializeComponent();
14. SubjectTasksStudent = subjectTasksStudent;
15. Test = test;
16. QuestionInd = 0;
17. QuestionMarks = new List<int>();
18. Test.StudentMarksList = new List<TestMark>();
19. if (Test.RandQuestionOrder)
20. NewOrder = Services.randomizeList(Test.Questions);
21. else
22. NewOrder = new List<TestQuestion>(Test.Questions);
23. }
24. private void ViewTest\_Load(object sender, EventArgs e) {
25. StartTime = DateTime.Now;
26. NextQ();
27. }
28. private void button1\_Click(object sender, EventArgs e) { // End test
29. TimeSpan TimeSpent = DateTime.Now - StartTime;
30. saveResult();
31. DialogResult dr = MessageBox.Show("Are you sure you want to end test?", "", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
32. if (dr == DialogResult.OK) {
33. TestMark tm;
34. if (Test.RandQuestionOrder)
35. tm = Services.derandomizeMarks(this, TimeSpent);
36. else
37. tm = new TestMark(SubjectTasksStudent.StudentMainMenu.Student.Username, QuestionMarks, TimeSpent);
38. Test.StudentMarksList.Add(tm);
39. Test.WriteToJson(SubjectTasksStudent.TestDir);
40. SubjectTasksStudent.Show();
41. Close();
42. }
43. }
44. private void button6\_Click(object sender, EventArgs e) { // Next
45. saveResult();
46. QuestionInd++;
47. NextQ();
48. }
49. private void NextQ() {
50. label1.Text = NewOrder[QuestionInd].Question;
51. Services.nextQuestion(this, richTextBox1);
52. if (QuestionInd == Test.Questions.Count - 1) {
53. button1.Visible = true;
54. button6.Visible = false;
55. }
56. }
57. private void saveResult() {
58. TestQuestion currentQ = NewOrder[QuestionInd];
59. int finalMark = 0;
60. if (currentQ.WrongAns.Count == 0) { // detailed answer
61. int diff = string.Compare(richTextBox1.Text, currentQ.RightAns[0]);
62. finalMark = diff >= 0 ? currentQ.Value - diff : 0;
63. richTextBox1.Visible = false;
64. } else {
65. if (currentQ.RightAns.Count == 1) { // only one right answer
66. List<RadioButton> allRadBtn = Services.getCollection<RadioButton>(this);
67. foreach (RadioButton rb in allRadBtn)
68. if (rb.Checked && currentQ.RightAns.Contains(rb.Text)) {
69. finalMark = currentQ.Value;
70. break;
71. }
72. foreach (RadioButton rb in allRadBtn)
73. Controls.Remove(rb);
74. } else if (currentQ.RightAns.Count > 1) { // multiple right answers
75. List<CheckBox> allChkBox = Services.getCollection<CheckBox>(this);
76. int checkedRightAnsCount = 0;
77. foreach (CheckBox cb in allChkBox)
78. if (cb.Checked && currentQ.RightAns.Contains(cb.Text))
79. checkedRightAnsCount++;
80. foreach (CheckBox cb in allChkBox)
81. Controls.Remove(cb);
82. finalMark = (currentQ.Value / currentQ.RightAns.Count) \* checkedRightAnsCount;
83. }
84. }
85. QuestionMarks.Add(finalMark);
86. }
87. }
88. }

# Appendix B. Class Diagram

@startuml

interface IUser {

Password : string <<get>> <<set>>

SecretAnswer : string <<get>> <<set>>

SecretQuestion : string <<get>> <<set>>

Username : string <<get>> <<set>>

}

class Teacher {

+ Teacher(username:string, password:string, secretQuestion:string, secretAnswer:string, subject:string)

+ Subject : string <<get>> <<set>>

+ Password : string <<get>> <<set>>

+ SecretAnswer : string <<get>> <<set>>

+ SecretQuestion : string <<get>> <<set>>

+ Username : string <<get>> <<set>>

}

IUser <|-- Teacher

class Student {

+ Student(username:string, password:string, secretQuestion:string, secretAnswer:string, group:string)

+ Student(username:string, password:string, secretQuestion:string, secretAnswer:string, group:string, studyTime:TimeSpan)

+ Group : string <<get>> <<set>>

+ Password : string <<get>> <<set>>

+ SecretAnswer : string <<get>> <<set>>

+ SecretQuestion : string <<get>> <<set>>

+ Username : string <<get>> <<set>>

}

IUser <|-- Student

Student --> "StudyTime" TimeSpan

@enduml

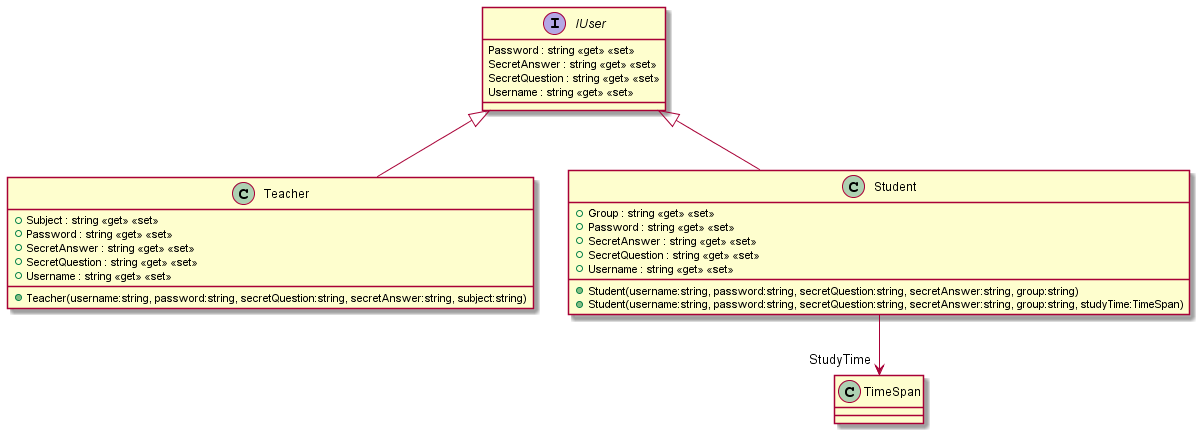


Image 1-B. Diagram of IUser, Teacher, and Student

@startuml

interface IGroup {

Name : string <<get>> <<set>>

Specialty : string <<get>> <<set>>

Year : string <<get>> <<set>>

addStudent(username:string) : bool

rmStudent(username:string) : bool

}

class "List`1"<string> {

}

IGroup --> "Students " "List`1"

IGroup --> "Subjects " "List`1"

class Group {

+ Group(specialty:string, year:string)

+ Group(grName:string)

+ Name : string <<get>> <<set>>

+ Specialty : string <<get>> <<set>>

+ Year : string <<get>> <<set>>

+ addStudent(username:string) : bool

+ rmStudent(username:string) : bool

}

IGroup <|-- Group

@enduml

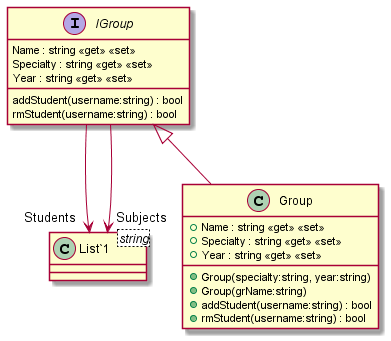


Image 2-B. Diagram of IGroup and Group

@startuml

interface ILecture {

Name : string <<get>> <<set>>

Semester : int <<get>> <<set>>

Text : string <<get>> <<set>>

addImg(fileName:string) : bool

WriteToJson(dir:string) : void

}

class "List`1"<string> {}

class "List`2"<TestMark> {}

ILecture --> "ImgList" "List`1"

ILecture --> "StudentMarksList" "List`2"

class Lecture {

+ Lecture(name:string, text:string, semester:int, imgList:List<string>)

+ Lecture()

+ Name : string <<get>> <<set>>

+ Text : string <<get>> <<set>>

+ Semester : int <<get>> <<set>>

+ addImg(fileName:string) : bool

+ WriteToJson(dir:string) : void

}

ILecture <|-- Lecture

@enduml

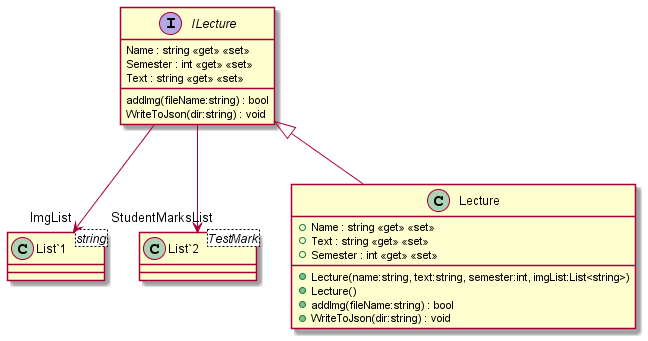


Image 3-B. Diagram of ILecture and Lecture

@startuml

interface ITest {

Name : string <<get>> <<set>>

RandQuestionOrder : bool <<get>> <<set>>

Semester : int <<get>> <<set>>

getStudentMark(studentUsrName:string) : int

maxScore() : int

WriteToJson(dir:string) : void

getQuestions() : List<string>

}

class "List`1"<TestMark> {}

class "List`2"<TestQuestion> {}

ITest --> "Questions" "List`2"

ITest --> "StudentMarksList" "List`1"

class Test {

+ Test(name:string, questions:List<TestQuestion>, studentMarksList:List<TestMark>, randQuestionOrder:bool, semester:int)

+ Test(name:string, questions:List<TestQuestion>, randQuestionOrder:bool, semester:int)

+ Test()

+ RandQuestionOrder : bool <<get>> <<set>>

+ Semester : int <<get>> <<set>>

+ Name : string <<get>> <<set>>

+ getStudentMark(studentUsrName:string) : int

+ maxScore() : int

+ WriteToJson(dir:string) : void

+ getQuestions() : List<string>

}

ITest <|-- Test

@enduml

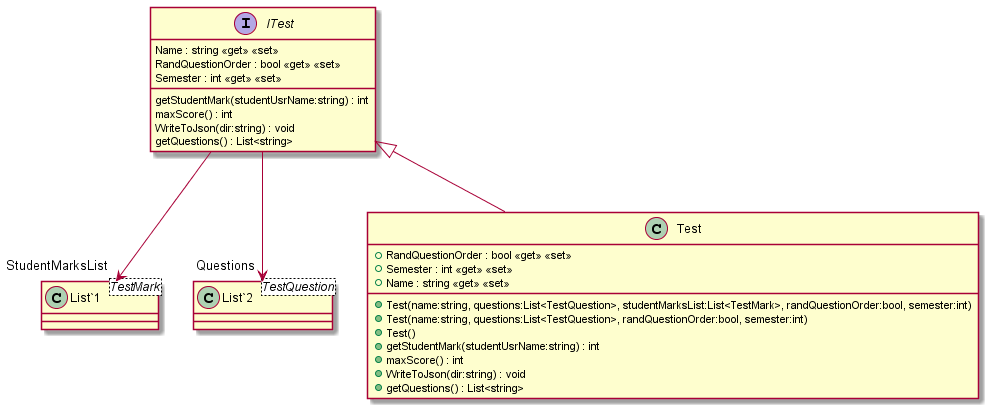


Image 4-B. Diagram of ITest and Test

@startuml

interface ITestQuestion {

Question : string <<get>> <<set>>

Value : int <<get>> <<set>>

}

class "List`1"<string> {

}

ITestQuestion --> "RightAns" "List`1"

ITestQuestion --> "WrongAns" "List`1"

class TestQuestion {

+ Value : int <<get>> <<set>> = 1

+ Question : string <<get>> <<set>>

}

ITestQuestion <|-- TestQuestion

@enduml

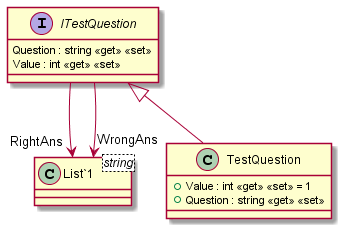


Image 5-B. Diagram of ITestQuestion and TestQuestion

@startuml

interface ITestMark {

StudentUsrName : string <<get>> <<set>>

getStudentMark(maxScore:int) : int

}

class "List`1"<int> {

}

ITestMark --> "Marks" "List`1"

ITestMark --> "TimeSpent" TimeSpan

class TestMark {

+ StudentUsrName : string <<get>> <<set>>

+ TestMark()

+ TestMark(studentUsrName:string, marks:List<int>, timeSpent:TimeSpan)

+ getStudentMark(maxScore:int) : int

}

ITestMark <|-- TestMark

@enduml

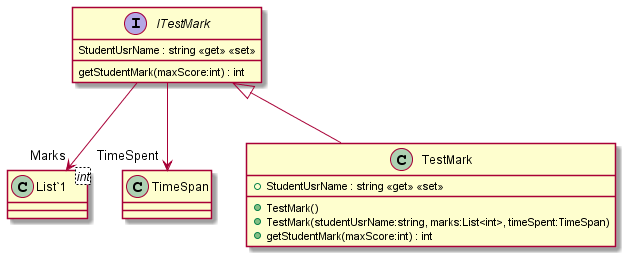


Image 6-B. Diagram of ITestMark and TestMark

@startuml

class Rules {

+ getSubjList(spec:string, year:string) : string[]

+ {static} getSubjList(grName:string) : List<string>

}

@enduml

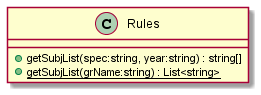


Image 7-B. Diagram of Rules

@startuml

class Services {

+ {static} getRand(start:int, end:int) : int

+ {static} GetCurrentSemester() : int

+ {static} getOrder(dir:string) : List<string>

+ {static} rewriteOrder(dir:string, newOrder:List<string>) : void

+ {static} appendToOrder(dir:string, nameToAppend:string) : void

+ {static} saveOrder(dir:string, dgv:DataGridView) : void

+ {static} createBtnList(x:int, y:int, btnTextList:List<string>, delFunc:btnOnClick) : List<Button>

+ {static} createRadioBtnList(x:int, y:int, btnTextList:List<string>) : List<RadioButton>

+ {static} createChkBoxList(x:int, y:int, btnTextList:List<string>) : List<CheckBox>

+ {static} getGroupList() : List<string>

+ {static} getGroupsWithSubj(subject:string) : List<string>

+ {static} deserializeObj(filePath:string) : T

+ {static} nextImg(currentImg:int, lecture:Lecture, pcb:PictureBox) : void

+ {static} previousImg(currentImg:int, lecture:Lecture, pcb:PictureBox) : void

+ {static} randomizeList(listToRandomize:List<T>) : List<T>

+ {static} nextQuestion(ViewTest:ViewTest, rtb:RichTextBox) : void

+ {static} getCollection(Form:Form) : List<T>

+ {static} derandomizeMarks(vt:ViewTest, timeSpent:TimeSpan) : TestMark

+ {static} openCP(groupInfo:GroupInfo) : void

+ {static} DGVCellContentClick(sender:object, e:DataGridViewCellEventArgs, colInd:int) : string

+ {static} fillDGV(dgw:DataGridView, nameList:List<string>, btnText:string) : void

+ {static} DGVMouseClick(dgv:DataGridView, e:MouseEventArgs, Rw:DataGridViewRow, RowIndexFromMouseDown:int) : void

+ {static} DGVDragEnter(dgv:DataGridView, e:DragEventArgs) : void

+ {static} DGVragDrop(dgv:DataGridView, e:DragEventArgs, Rw:DataGridViewRow, RowIndexFromMouseDown:int) : void

}

@enduml



Image 8-B. Diagram of Services

@startuml

class RegistrationServices {

+ {static} validateUsrName(usrName:string) : bool

+ {static} validatePass(password:string) : bool

+ {static} passMatch(password:string, repeatPass:string) : bool

}

@enduml

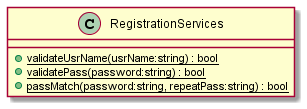


Image 9-B. Diagram of RegistrationServices

@startuml

class Form1 <<partial>> {

+ Form1()

}

Form <|-- Form1

@enduml

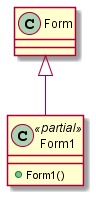


Image 10-B. Diagram of Form1

@startuml

class Registration <<partial>> {

+ Registration(form1:Form1)

}

Form <|-- Registration

@enduml

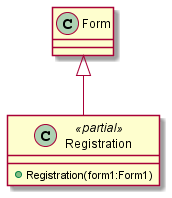


Image 11-B. Diagram of Registration

@startuml

class ForgotPass <<partial>> {

+ ForgotPass(form1:Form1)

}

Form <|-- ForgotPass

@enduml

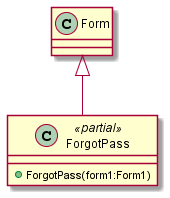


Image 12-B. Diagram of ForgotPass

@startuml

class TeacherMainMenu <<partial>> {

+ TeacherMainMenu(teacher:Teacher, form1:Form1)

}

Form <|-- TeacherMainMenu

TeacherMainMenu --> "Form1" Form1

TeacherMainMenu --> "Teacher" Teacher

@enduml

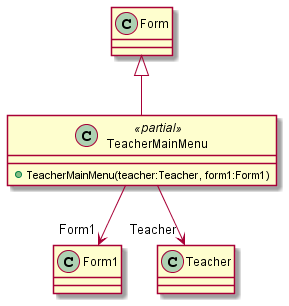


Image 13-B. Diagram of TeacherMainMenu

@startuml

class GroupInfo <<partial>> {

+ LectDir : string <<get>> <<set>>

+ TestDir : string <<get>> <<set>>

+ GroupInfo(grName:string, teacherMM:TeacherMainMenu)

}

Form <|-- GroupInfo

GroupInfo --> "CurGr" Group

GroupInfo --> "TeacherMM" TeacherMainMenu

@enduml

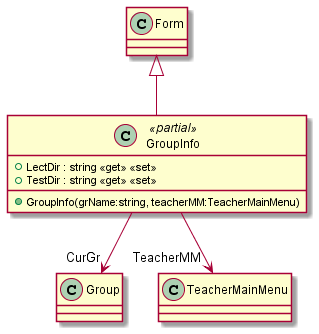


Image 14-B. Diagram of GroupInfo

@startuml Add\_Lecture

class Add\_Lecture <<partial>> {

+ Changed : bool <<get>> <<set>>

+ CourseProject : bool <<get>> <<set>>

+ EditMode : bool <<get>> <<set>>

+ CurrentImg : int <<get>> <<set>>

+ Dir : string <<get>> <<set>>

+ Add\_Lecture(grInfoForm:GroupInfo, courseProject:bool)

+ Add\_Lecture(grInfoForm:GroupInfo, lecture:Lecture, courseProject:bool)

}

Form <|-- Add\_Lecture

Add\_Lecture --> "GrInfoForm" GroupInfo

Add\_Lecture --> "Lecture" Lecture

@enduml

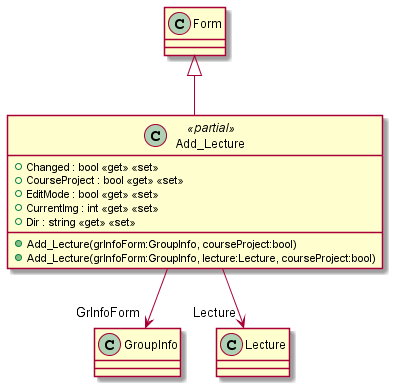


Image 15-B. Diagram of Add\_Lecture

@startuml

class AddTest <<partial>> {

+ Changed : bool <<get>> <<set>>

+ EditMode : bool <<get>> <<set>>

+ SavedToOrder : bool <<get>> <<set>>

+ Dir : string <<get>> <<set>>

+ OldName : string <<get>> <<set>>

+ AddTest(grInfoForm:GroupInfo)

+ AddTest(grInfoForm:GroupInfo, test:Test, savedToOrder:bool)

}

Form <|-- AddTest

AddTest --> "GrInfoForm" GroupInfo

AddTest --> "Test" Test

class Add\_Test\_Question <<partial>> {

+ OldQText : string <<get>> <<set>>

+ EditMode : bool <<get>> <<set>>

+ Changed : bool <<get>> <<set>>

+ Add\_Test\_Question(addTestForm:AddTest)

+ Add\_Test\_Question(addTestForm:AddTest, question:TestQuestion)

}

Form <|-- Add\_Test\_Question

Add\_Test\_Question --> "AddTestForm" AddTest

@enduml

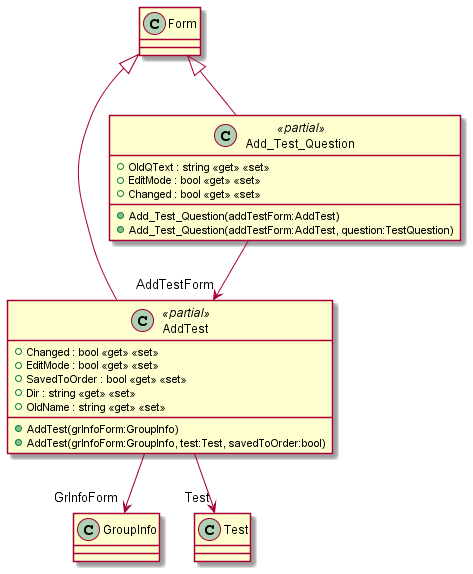


Image 16-B. Diagram of AddTest and Add\_Test\_Question

@startuml

class StudentStatistics <<partial>> {

+ StudentUsrName : string <<get>> <<set>>

+ StudentStatistics(groupInfoForm:GroupInfo, studentUsrName:string)

}

Form <|-- StudentStatistics

StudentStatistics --> "GroupInfoForm" GroupInfo

StudentStatistics --> "TotalTime" TimeSpan

@enduml

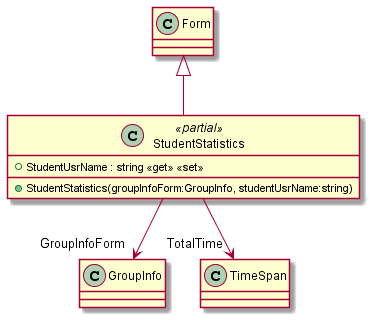


Image 17-B. Diagram of StudentStatistics

@startuml

class StudentMainMenu <<partial>> {

+ StudentMainMenu(student:Student, form1:Form1)

}

Form <|-- StudentMainMenu

StudentMainMenu --> "Student" Student

@enduml

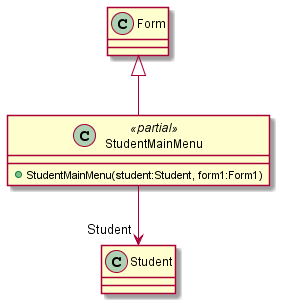


Image 18-B. Diagram of StudentMainMenu

@startuml

class SubjectTasksStudent <<partial>> {

+ LectDir : string <<get>> <<set>>

+ Subject : string <<get>> <<set>>

+ TestDir : string <<get>> <<set>>

+ SubjectTasksStudent(subject:string, studentMainMenu:StudentMainMenu)

}

Form <|-- SubjectTasksStudent

SubjectTasksStudent --> "StudentMainMenu" StudentMainMenu

@enduml

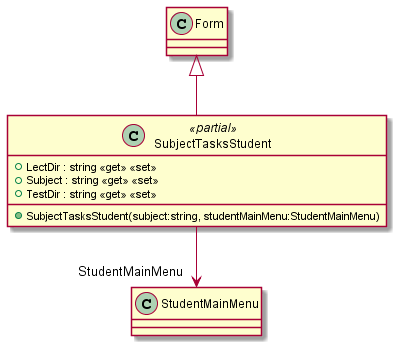


Image 19-B. Diagram of SubjectTasksStudent

@startuml

class ViewLecture <<partial>> {

+ CurrentImg : int <<get>> <<set>>

+ ViewLecture(subjectTasksStudent:SubjectTasksStudent, lecture:Lecture)

}

Form <|-- ViewLecture

ViewLecture --> "StartTime" DateTime

ViewLecture --> "Lecture" Lecture

ViewLecture --> "SubjectTasksStudent" SubjectTasksStudent

@enduml

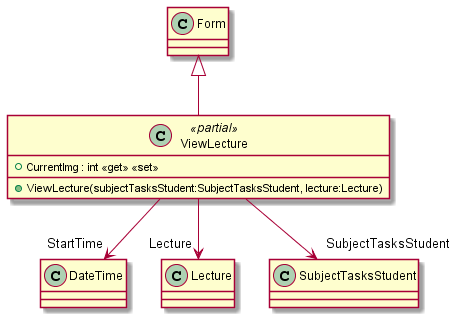


Image 20-B. Diagram of ViewLecture

@startuml

class ViewTest <<partial>> {

+ QuestionInd : int <<get>> <<set>>

+ ViewTest(subjectTasksStudent:SubjectTasksStudent, test:Test)

}

class "List`1"<TestQuestion> {}

class "List`2"<int> {}

Form <|-- ViewTest

ViewTest --> "StartTime" DateTime

ViewTest --> "QuestionMarks" "List`2"

ViewTest --> "NewOrder" "List`1"

ViewTest --> "SubjectTasksStudent" SubjectTasksStudent

ViewTest --> "Test" Test

@enduml

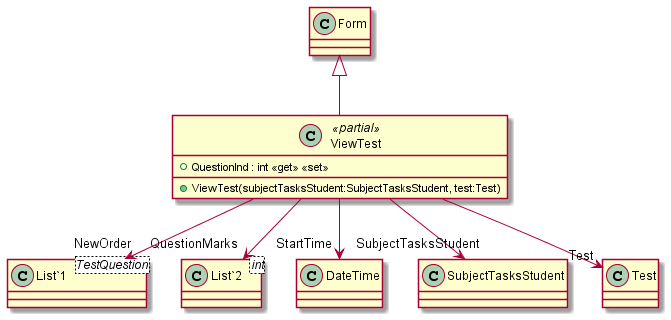


Image 21-B. Diagram of ViewTest

# Appendix C. Form Screenshots