

Ministry of Education and Science of Ukraine
Kharkiv National University of Radio Electronics

Faculty _____ Computer Science _____

Department _____ Systems Engineering _____

COURSE WORK

EXPLANTORY NOTE

discipline _____ Object-oriented programming _____
(name of discipline)

theme: _____ Development of object-oriented software system “Gradebook” _____
(theme of work)

Head _____ Senior Lecturer Zhernova P. E. _____
(signature, date, position, surname, initials)

Student _____ CST-20-1 _____ Hrobovyi D. V. _____
(group, signature, date, surname, initials)

The work is protected with an assessment « _____ »
« _____ » _____ 2021

Commission:

_____ As. Prof. of Dep. SysEng Reshetnik V. M. _____
(signature, position, surname, initials)

_____ Sen. Lect. of Dep. SysEng Zhernova P. E. _____
(signature, position, surname, initials)

_____ Assist. Lect. of Dep. SysEng Chorna O. S. _____
(signature, position, surname, initials)

Kharkiv 2021

Kharkiv National University of Radio Electronics

Faculty Computer Science

Department Systems Engineering

Educational program Computer Science and Technology

Course 1 group CST-20-1 **term** 2

TASK

for course work

for student Hrobovyi Danylo Vitaliyovych

(surname, name, patronymic)

1. Theme of work: Development of object-oriented software system “Gradebook”

2. The deadline for the student to complete the work 31.05.2021

3. Initial data to the project: Develop an object-oriented software system “Gradebook” with a hierarchy of classes and support for polymorphism in the C++ programming language. The software system performs the functions defined in the variant of the task: 20 (this information can be changed and clarified during the work). Operating system – Windows 7 or higher, software: MS Visual Studio integrated environment, UML-diagram editor Software Ideas Modeler. Methodical support – methodical instructions to course work.

4. Contents of the explanatory note (list of issues to be developed): to analyze the subject area and identify entities, objects, their attributes and functions; formulate and draw up requirements for the software system; develop an object model of the subject area and form a dictionary; conduct UML modeling, develop a class diagram; to implement the designed software system in C++ language and to describe it; develop a user interface; perform testing of the developed program; to prepare in accordance with GOST 19.401-78 program document “Program text”.

5. List of graphic material: Scheme of object model, class diagram, algorithms, examples of screen forms.

6. Date of issue of the task: 01.04.2020

Head Zhernova Polina Yevheniyivna

(Signature) (surname, name, patronymic)

Student Hrobovyi Danylo Vitaliyovych

(Signature) (surname, name, patronymic)

CALENDAR PLAN

№	Name of stages of the course project	Deadline	Note
1	Issuance of the topic, coordination and approval of the topic.	01.04.2021	done
2	Subject area analysis.	04.04.2021	done
3	Formulation of requirements to the program, development of the system dictionary.	08.04.2021	done
4	Development of a class diagram.	12.04.2021	done
5	Development of base class and derived classes.	16.04.2021	done
6	Development of the program interface in the form of a text menu.	22.04.2021	done
7	Development of member functions of assigned classes for the program to work. Functional check of functions.	26.04.2021	done
8	Development of member functions of classes designed to remove / add elements. Functional check of functions.	30.04.2021	done
9	Development of member functions of classes designed for sorting, searching, maintaining / outputting elements. Functional check of functions.	05.05.2021	done
10	Development of the main function main(). System health check.	15.05.2021	done
11	Testing of the developed software.	20.05.2021	done
12	Preparation of an explanatory note and its appendices.	31.05.2021	done

Head _____ **Zhernova Polina Yevheniyivna**
 (Signature) (surname, name, patronymic)

Student _____ **Hrobovyi Danylo Vitaliyovych**
 (Signature) (surname, name, patronymic)

« ____ » _____ 2021.

ABSTRACT

Explanatory note to the course work: 47 p., 0 tables, 23 fig., 1 appendix, 5 sources of information.

CLASS, POLYMORPHIC CLUSTER, VIRTUAL FUNCTION, OBJECT-ORIENTED DESIGN, SOFTWARE SYSTEM, USER INTERFACE

The purpose of the work is to develop a software system that simulates the functioning of the gradebook, using the paradigm of object-oriented design and programming.

The object of development is an object model of the gradebook operation.

The subject of development is information technologies and software methods of object-oriented software system development.

The development uses the object-oriented capabilities of the C++ programming language and the integrated development environment MS Visual Studio 2019.

The analysis of the subject area is carried out, the main tasks of the system are defined, the description of input and output information is developed, the hierarchical class model of system is developed, the diagram of classes is executed, the software realizing the basic functions is developed: reading and processing entered information, write to file.

The program provides quick receipt of the necessary information and increase the efficiency of the teacher and security of data.

Scope is support for the electronic version of the log of information of all exams of the student.

CONTENTS

Abbreviations and explanations	6
Introduction	7
1 Analysis of the subject area	8
1.1 Subject area analysis	9
1.1 Formulation of the problem	9
2 Description of the software implementation	11
2.1 Description of the class "Person"	11
2.2 Description of the class "Teacher"	12
2.3 Description of the class "Student"	13
2.4 Description of the class "Exam"	14
2.5 Description of the hierarchy of classes "Person", "Teacher", "Student" and "Exam"	15
2.6 Description of the class "Gradebook"	16
2.7 Description of the class "Iterator"	17
2.8 Relation of classes: Gradebook, Iterator	18
2.4 Description of the class "Menu"	19
2.9 Hierarchy of classes: Menu, MenuItem, Timer and Keys enum class	21
2.11 Hierarchy of all application	22
3 Software Implementation	23
3.1 Description of structure and algorithms	23
3.2 User interface description	24
Conclusions	32
References	33
Appendix A	34

ABBREVIATIONS AND EXPLANATIONS

- Getter — function that gives the value of a closed attribute
- Setter — function that sets the value of a closed attribute
- OOP — object-oriented programming
- UML — is a general-purpose, developmental, modeling language that is intended to provide a standard way to visualize the design of a system

INTRODUCTION

In the current reality, the education system is overloaded with paperwork, all data are stored in huge archives and data retrieval, as well as the cost of maintaining all this infrastructure leaves much to be desired. This has been the case since the advent of writing, but nowadays there is an alternative in the form of more efficient ways of storing and processing data, known as information technology. Information technology allows you to store huge amounts of data on small and inexpensive storage media, and data processing, i.e. searching, sorting, etc. will no longer require so much time and effort.

But to solve business problems, which include data processing, it is necessary to create an application with the required functionality, and for this, it is necessary to use programming. Programming is the process of creating programs by writing program code using a programming language. There are many programming paradigms that increase the scalability, productivity and maintainability of program code.

Nowadays, the most widespread programming paradigm is OOP. Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.[1]

Some of the benefits of object-oriented programming include:

1. Increased software development productivity: Object-oriented programming is modular because it provides a division of responsibilities when developing programs based on objects. It is also extensible because objects can be extended to include new attributes and behaviors. Objects can also be reused within applications. Because of these three factors - modularity, extensibility, and reusability - object-oriented programming provides better software development performance than traditional procedure-based programming methods.

2. Improved maintainability of software: For the reasons mentioned above, object-oriented software is also easier to maintain. Because the design is modular, parts of the system can be updated when problems arise without the need for extensive changes.

3. Faster development: Reuse allows faster development. Object-oriented programming languages come with rich object libraries, and code developed during projects can also be reused in future projects.

4. Reduced development costs: Reusing software also reduces development costs. Generally, more effort is put into object-oriented analysis and design, which reduces the overall cost of development.

5. Better software: Faster software development and lower development costs allow more time and resources to be used to test the software. Although quality depends on the experience of the teams, object-oriented programming generally results in better software.[2]

The object-oriented approach, compared to others, really makes life easier for a programmer and gives the opportunity to implement massive programs that can be worked on by several programmers. All of this based on the above, the goal was set: to implement the program using an object-oriented approach using the C ++ programming language.[3]

1 ANALYSIS OF THE SUBJECT AREA

1.1 Subject area analysis

In this course work was chosen the variant of task №20 - "Gradebook", we have a task to develop a program that implement gradebook. And during the work we need to create software to fill, exam fields of gradebook, print them, save them to file and load them from file, using C ++ language skills and knowledge of OOP.

In order to do the work, it is necessary to consider theoretical information from the program area and from a technical point of view. Namely: create several classes to describe objects, methods to manage data, implement inheritance, polymorphic cluster, overload operations, etc.

The relevance of the program lies in its application to educational institutions. Simplifying the grading process will increase the effectiveness of teachers in distance learning.

1.2 Formulation of the problem

Formulation the problem requires analyzing the subject area and defining entities, objects, their attributes and functions. The development of an object-oriented software system "Gradebook" should go through some steps, such as:

- Create a Person class with a second name field, define initialization, copy, virtual destructor, getters and setters for fields of this class, constructors to change and read field values of this class. Overload object assignment operation =, input >> and output << operations.
- Create Teacher class with post field. Define input, output and assignment operators, constructors, destructor, getters and setters to change and read the values in the fields of this class, override virtual methods.
- Create Student class with group field. Define input, output and assignment operators, constructors, destructor, getters and setters to change and read field values of the class, override virtual methods.
- In the class hierarchy, construct a polymorphic cluster based on virtual methods for outputting class data to the console, for inputting data from the console to the class, and a method for converting data to string format. Demonstrate a late binding mechanism.
- Create an Exam class with fields: subject name, hours number, date, and mark. Define operators for input, output and assignment, constructors,

destructor, getters and setters for changing and reading values of fields of this class, override virtual methods.

- Create a Gradebook class with fields: exams – a container to store an array of exams and size – to display the current size of the container. Define operators for input, output and assignment, constructors, destructor.
- Create Iterator class to work with an array of objects.

2 DESCRIPTION OF THE SOFTWARE IMPLEMENTATION

2.1 Abstract base class Person

This class is created as the base class for Teacher and Student classes. Class Person contains the secondName field of string type, which contains the second name of the person, and getter and setter for this field. It also, defines 3 pure virtual functions: print – which is used to print class data to the console, fill – which is used to fill the class and toString – which converts class data to a string. The class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator, a move assignment operator and input and output operators. UML diagram of the class can be seen in the figure 2.1.1.

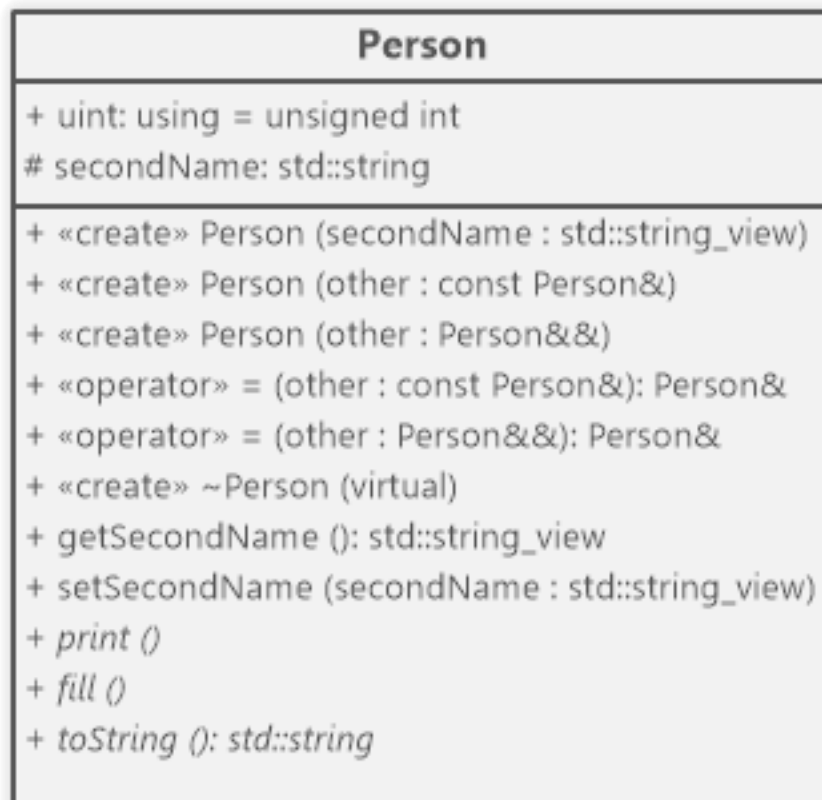


Figure 2.1.1 – Diagram of the class Person

2.2 Class Teacher

The Teacher class inherits the Person class, which provides it with a private field secondName. It also has its own private field post of type string, which contains the teacher's post title, as well as a getter and setter for that field. The Teacher class also overrides the virtual functions: print, fill, and toString. These functions allow

you to print Teacher class data to the console, fill it, and convert it to string format, respectively. The class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator, a move assignment operator and input and output operators. UML diagram of the class can be seen in the figure 2.2.1.

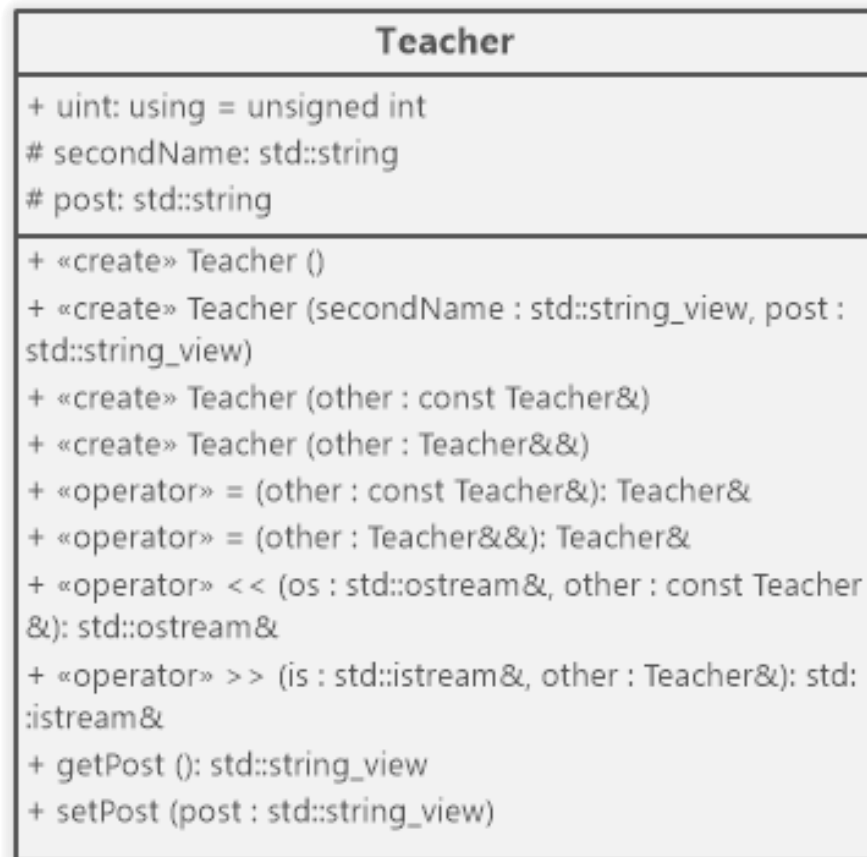


Figure 2.2.1 – Diagram of the class Teacher

2.3 Class Student

The Student class inherits the Person class, which provides it with a private field secondName. It also has its own private field group of type unsigned int, which contains the student's group number, as well as a getter and setter for that field. The Student class also overrides the virtual functions: print, fill, and toString. These functions allow you to print Student class data to the console, fill it, and convert it to string format, respectively. The class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator, a move assignment operator and input and output operators. UML diagram of the class can be seen in the figure 2.3.1.

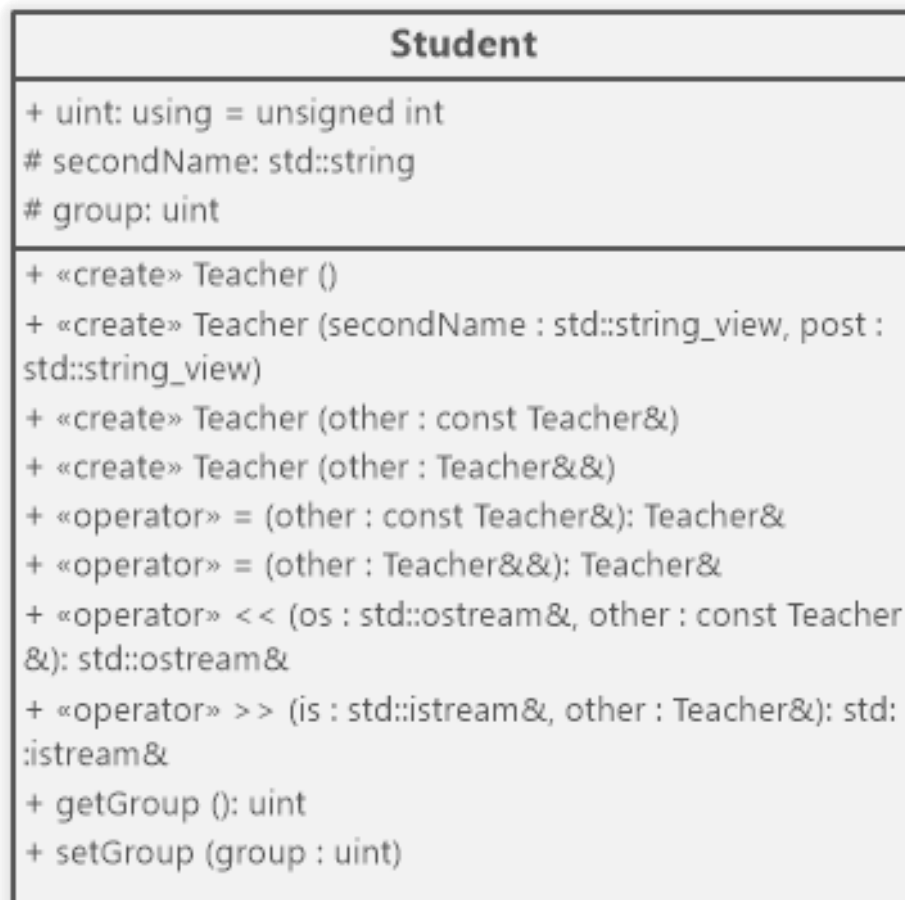


Figure 2.3.1 – Diagram of the class Student

2.4 Class Exam

The Exam class inherits Student and Teacher classes, which provides it with a private field secondName from each of them, closed field post from Teacher class and another closed field group from Student class. It also has its own private fields: subjectName of type string, hoursNumber of type unsigned int, date of type tm from C++ standart library and mark of type unsigned int, as well as a getters and setters for each of this fields. The Exam class also overrides the virtual functions: print, fill, and toString. These functions allow you to print Exam class data to the console, fill it, and convert it to string format, respectively. The class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator, a move assignment operator and input and output operators. UML diagram of the class can be seen in the figure 2.4.1.

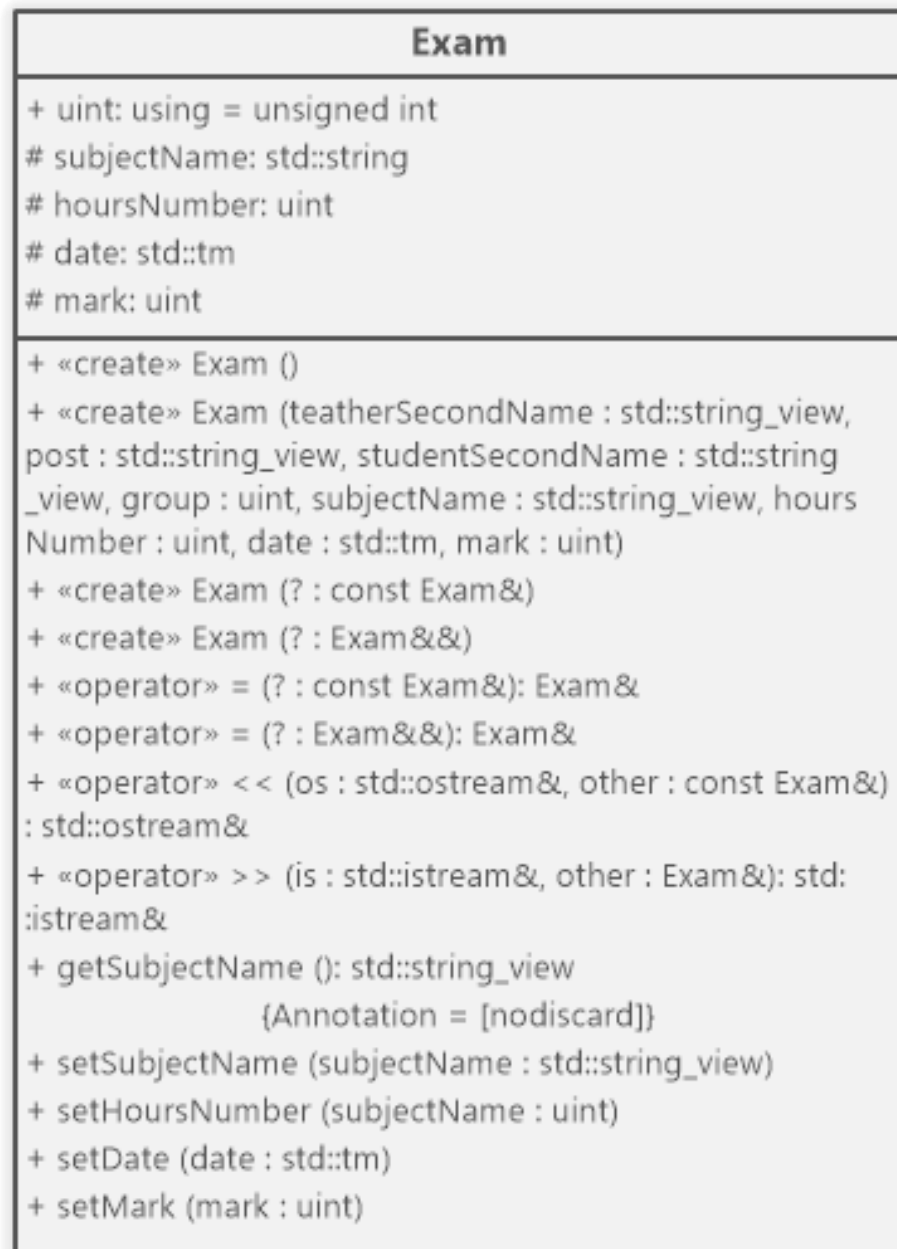


Figure 2.4.1 – Diagram of the class Exam

2.5 Hierarchy of classes: Person, Teacher, Student, Exam

The relation of application classes Person, Teacher, Student and Exam can be seen in Figure 2.8.1.

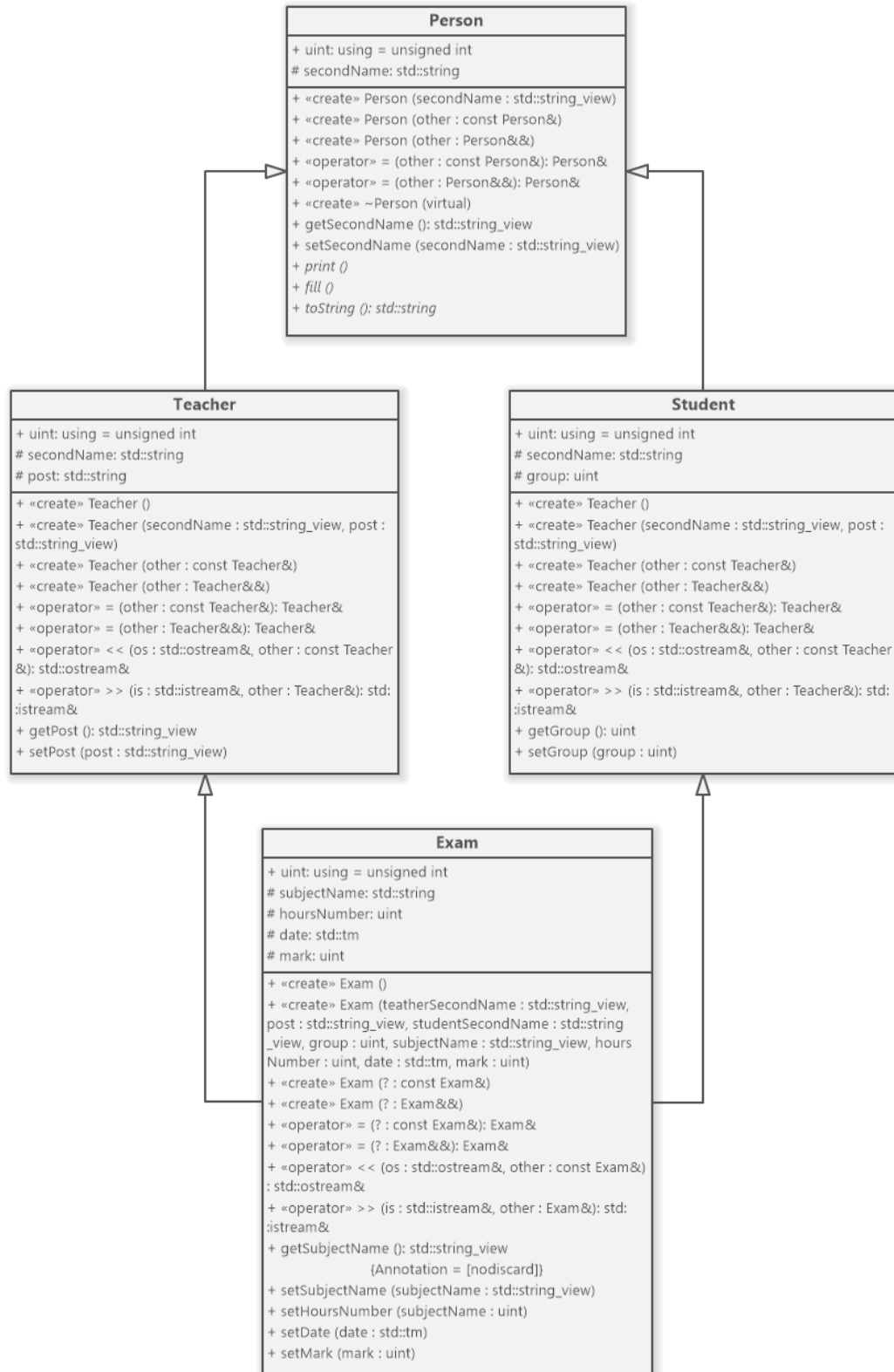


Figure 2.5.1 – Diagram of the hierarchy of classes Person, Teacher, Student and Gradebook

2.6 Class Gradebook

The Gradebook class has its own closed fields: exams – a container to store an array of exams and size – to display the current size of the container. The Gradebook class also has function print. Which allows printing to the console Exam class instances contained in exams field. The Gradebook class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator and a move assignment operator. Also, class implements push_back and pop_back methods to add and remove data respectively, a method to check if the container is empty – empty, the clean method to clean the container class and the size method that returns the actual size of the container, also, save and load methods to save to and load from the file respectively, as well as print method to output class data to the console. UML diagram of the class can be seen in the figure 2.6.1.

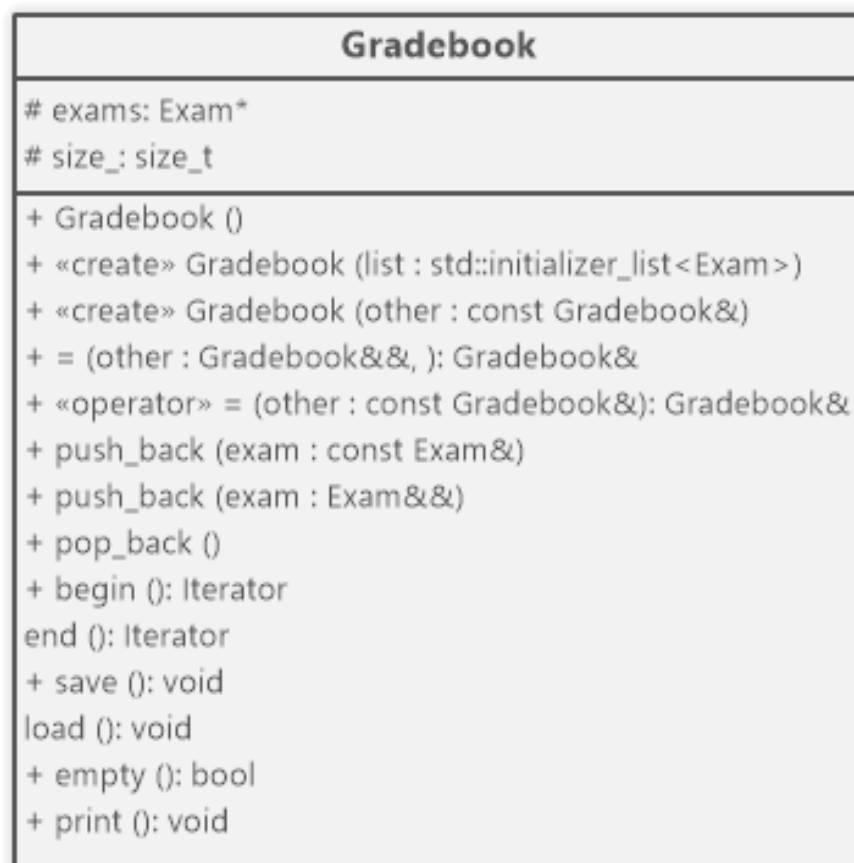


Figure 2.6.1 – Diagram of the class Gradebook

2.7 Class Iterator

Also, Gradebook class contains Iterator class which is used for iterating over array of exams. The Iterator class has its own closed field: `p` – a pointer to current element. The Gradebook class also has function `print`. The Iterator class also contains suffix increment operator, postfix increment operator, suffix decrement operator, postfix decrement operator, equality operator, inequality operator and dereference operator, which returns value of current element of container. UML diagram of the class can be seen in the figure 2.7.1.

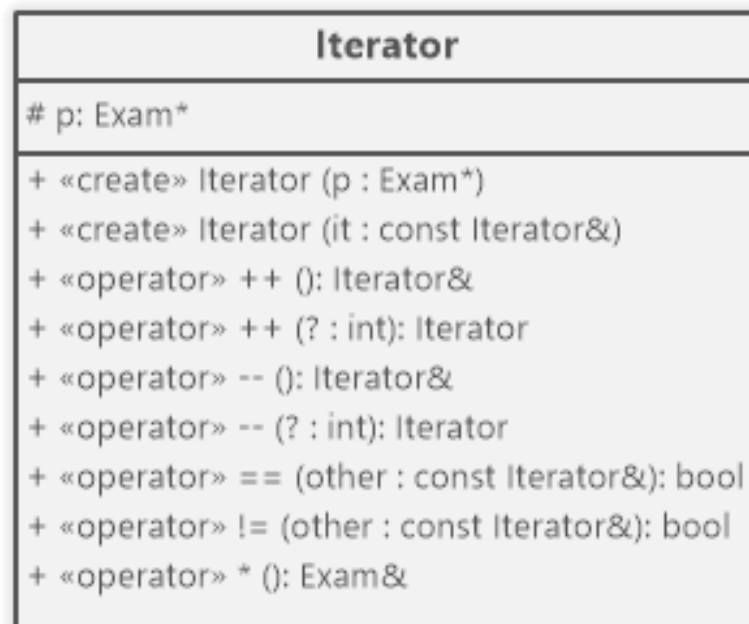


Figure 2.7.1 – Diagram of the class Iterator

2.8 Relation of classes: Gradebook, Iterator

The relation of application classes Gradebook and Iterator can be seen in Figure 2.8.1.

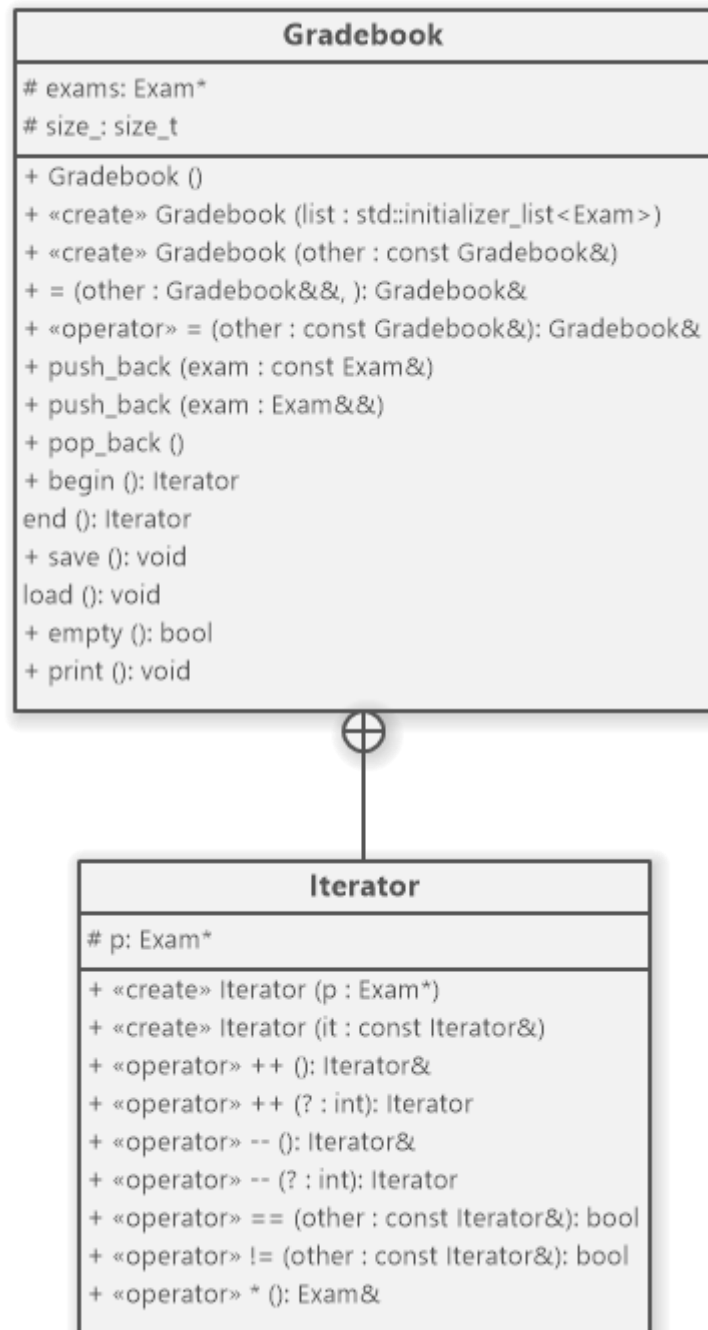


Figure 2.8.1 – Diagram of the relation of classes Gradebook and Iterator

2.9 Class Menu

The Menu class has closed fields: timer of custom type Timer, lock of mutex type used for managing threads work, hConsole of type HANDLE from Win32 API, and position of type int, menuName of type string, which contains name of menu, tooltip of string type which is used for storing tooltip string of menu and menuItems – vector of MenuItem type objects.

The Menu class contains methods:

- ChangeName, which changes the menuName field;
- AddItem, which creates item of menu;
- GetInput, which starts capturing user input; Initialize, which prepare console for menu output;
- SelectItem method which takes argument of int type and selects item of menu of this number;
- SetCursorToEnd method sets console cursor to the end of menu;
- ShowMenu method prints whole menu to the console;
- ShowTime method, which prints time to the console;
- ShowTooltip method, which prints tooltip to the console;
- ShowItems method, which prints items to the console;
- ShowCursor method, which hides or shows console cursor;
- CheckTime method which takes argument of custom Timer type and checks time for menu's timer

The class also contains a copy constructor and a move constructor to create based on already created instances of the class, a constructor with parameters as well as a copy assignment operator, a move assignment operator and input and output operators. UML diagram of the class can be seen in the figure 2.9.1.

The uniqueness of this menu is to display the time, which is updated regardless of the main user input stream, that is, the timer is calculated and displayed in a separate thread from the main input stream, while not conflicting with it.

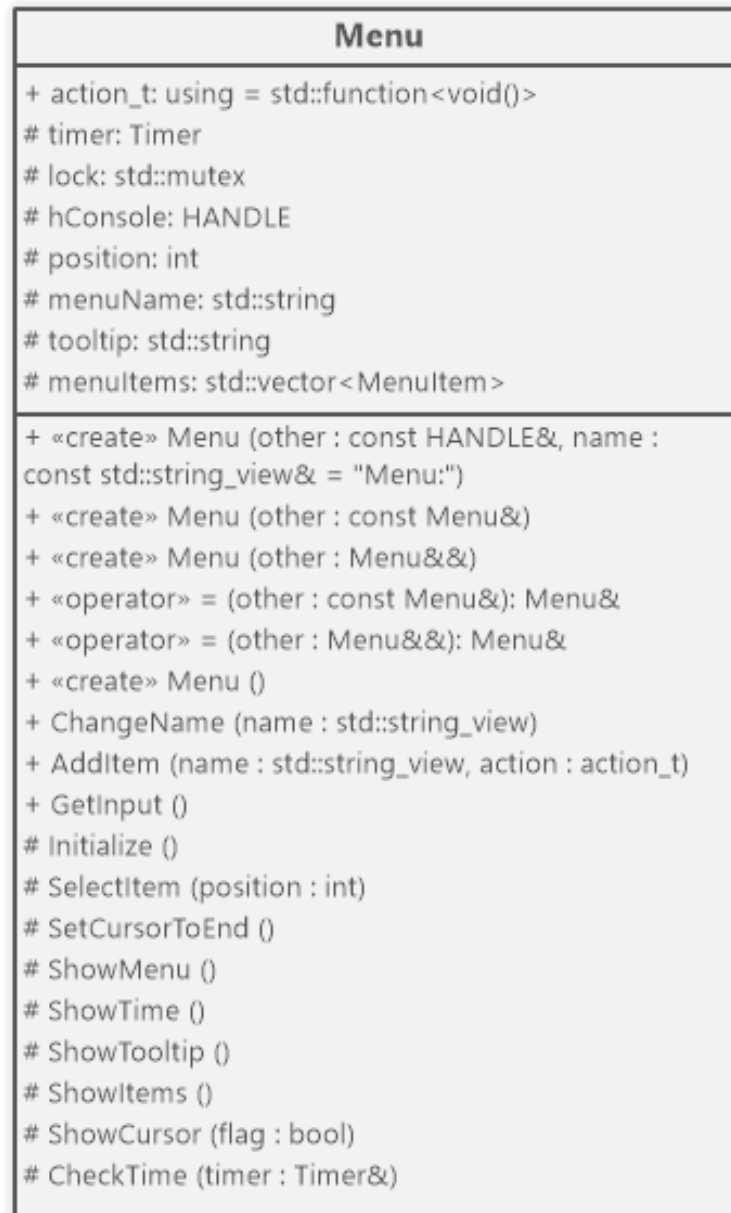


Figure 2.9.1 – Diagram of the hierarchy of classes Menu

2.10 Hierarchy of classes: Menu, MenuItem, Timer and Keys enum class

The hierarchy of application classes Menu, MenuItem, Timer and Keys enum class can be seen in Figure 2.11.1

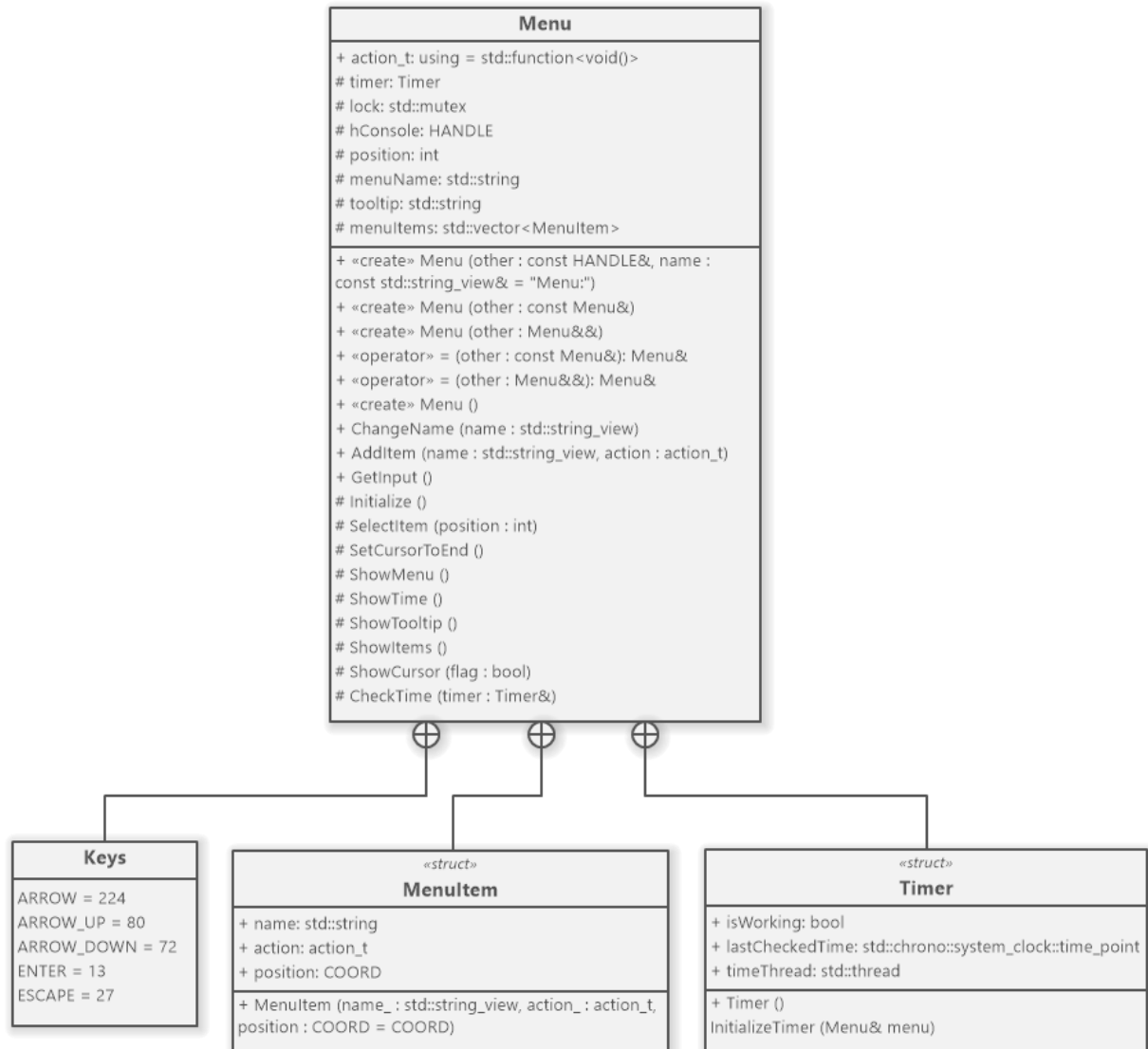


Figure 2.10.1 – Diagram of the hierarchy of classes Menu, MenuItem, Timer

2.11 Hierarchy of all application

The hierarchy of all application classes can be seen in Figure 2.11.1

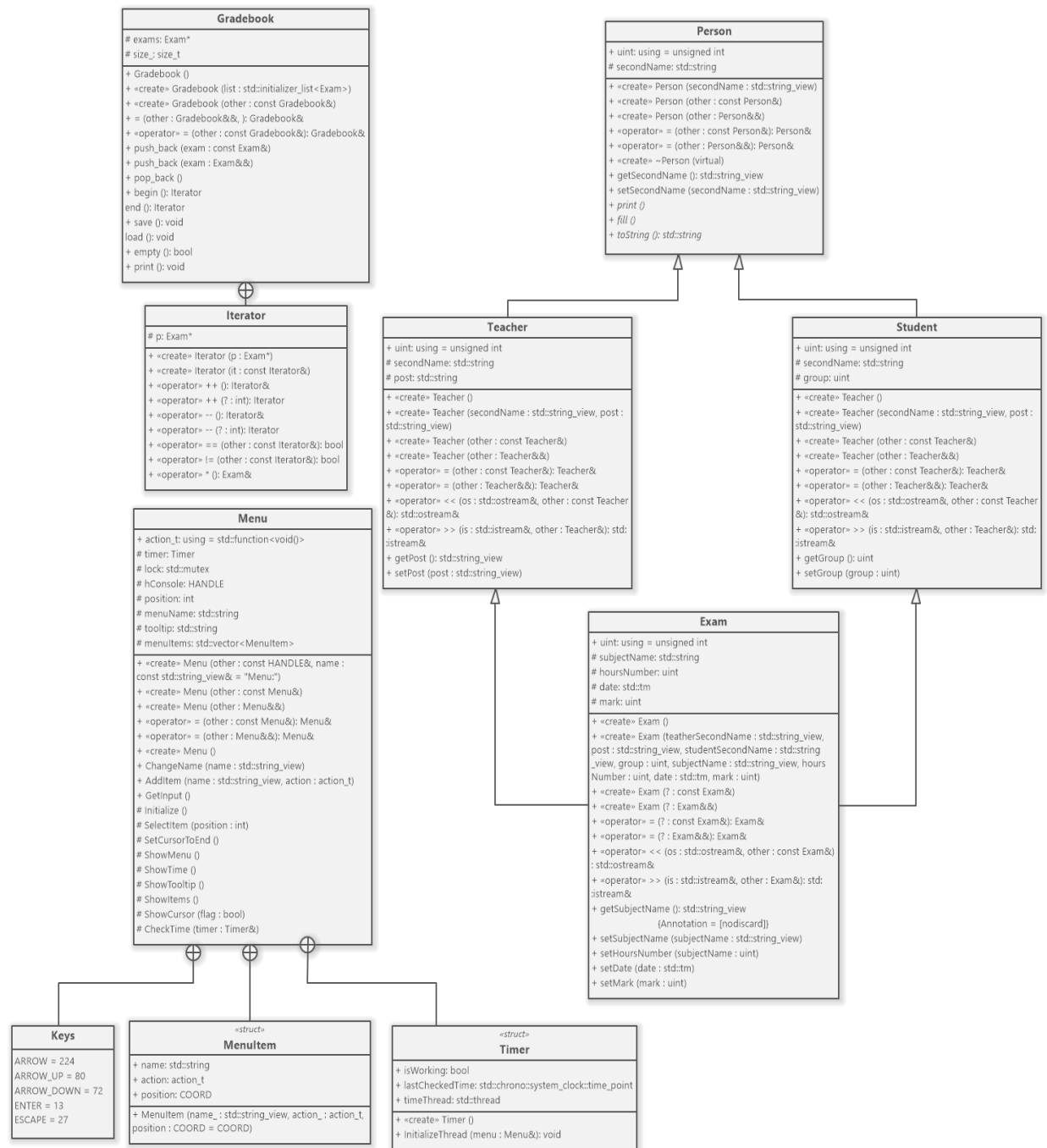


Figure 2.11.1 – Diagram of the hierarchy of all application classes

3 SOFTWARE IMPLEMENTATION

3.1 Description of structure and algorithms

The program contains 7 files of type .cpp and 6 header files that contains: pairs of files that describe the structure and content of classes (person.h, person.cpp, teacher.h, teacher.cpp, student.h, student.cpp, exam.h, exam.cpp, gradebook.h, gradebook.cpp, menu.h, menu.cpp) the main file that launches the program, describes the program menu.

Firstly, I created a project and the basic class – Person. Specified the required fields, and also defined the copy and move constructors, destructors, and methods for setting and getting the field values of this class.

The Teacher class is then created in a similar way, but it inherits from the previous, so Teacher objects have fields and methods that are described in the Person class.

Same way Student class was created. It is also inherited from Person class and has its own fields and methods, like field group and getter and setter for this filed.

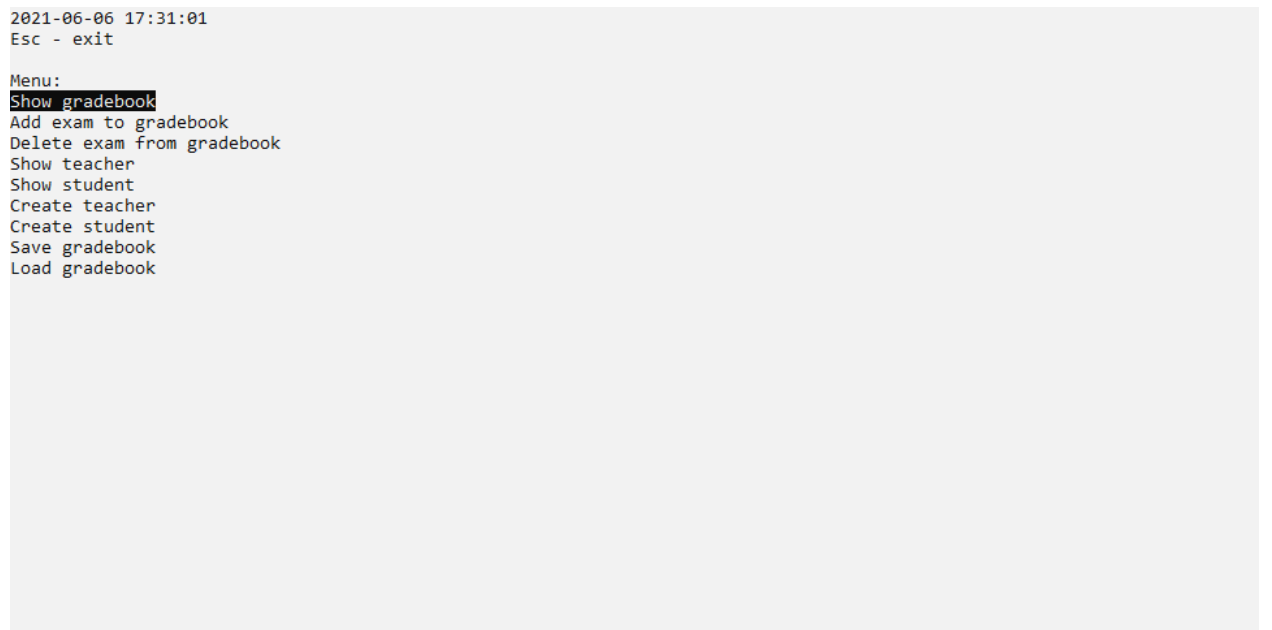
The Exam class was also created. The Exam class is derived from the Person and Student classes, it combines both the fields and methods of the Person class and the Student class, and also has its own fields and methods responsible for the name of the subject, the date of the exam and its grade.

The Gradebook class was also created as a container for the exam data type, the Iterator class was created for convenient interaction with the Gradebook class, the Gradebook class implements push_back and pop_back methods to add and remove data respectively, a method to check the container for empty, the clean method to clean the container class and the size method that returns the actual size of the container, as well as print method to output class data to the console.

The Menu class was created to simplify user interaction with the program. The Menu class provides an interface for creating menu items and assigning each item a certain action. The action, which is passed when creating a menu item, is transferred to the MenuItem class and stored there in the form of `std::function<void()>` from the `<functional>` library. To draw menu items, we use the console handler from the Win32 API. Also, we added an extra functionality to the menu in the form of a timer which was moved to another thread, to avoid conflicts when working with the console, we created mutex, which blocks one of our threads when drawing the time, or input operations, etc.

3.2 Interface description

The developed program has a simple and intuitive interface. Select menu items by pressing the up and down arrows on the keyboard, respectively. To select the desired menu item, select it and press the Enter key. The presence of interaction with files allows you to create lists of exams and save it to file or load it from it. The software application can be used in higher education institutions or used in the development of information systems in the future. On the figure 3.2.1 menu interface could be seen.




```
2021-06-06 17:31:01
Esc - exit

Menu:
Show gradebook
Add exam to gradebook
Delete exam from gradebook
Show teacher
Show student
Create teacher
Create student
Save gradebook
Load gradebook
```

Figure 3.2.1 – Menu interface


If we select the menu item "Show gradebook" when it is empty, our program will throw an exception and catch it with a message, as shown in Figure 3.2.2



```
Gradebook is empty
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.2 – Show gradebook menu item

Also, we can fill out our class using the "Add exam to gradebook" menu item, as we can see on the figure 3.2.3



```
Please enter exam data

Please enter teacher's data:
Second name: Ivanov
Post: Professor

Please enter student's data:
Second name: Hrobovyi
Group: 1

Subject data:
Subject name: Physics
Hours number: 20

Exam:
Date(format: 10:45-9.05.2021): 11:45-10.6.2021
Mark: 100
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.3 – Exam data input

If we select the menu item "Show gradebook" when it is not empty, our program will print stored in Gradebook object data, as shown in Figure 3.2.4

```
Teacher
Second name: Ivanov
Post: Professor

Student
Second name: Hrobovyi
Group: 1

Subject
Subject name: Physics
Hours number: 20

Exam
Date: 11:45 10.6.2020
Mark: 100

-----
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.4 – Show gradebook with data stored in it

We also can save and load our data from gradebook. Firstly, we have to fill our gradebook with some data, as in the figure 3.2.5.

```
Teacher
Second name: Vasiliev
Post: Professor

Student
Second name: Hrobovyi
Group: 1

Subject
Subject name: Calculus
Hours number: 20

Exam
Date: 10:15 7.7.2020
Mark: 98

-----
Teacher
Second name: Ivanov
Post: Professor

Student
Second name: Hrobovyi
Group: 1

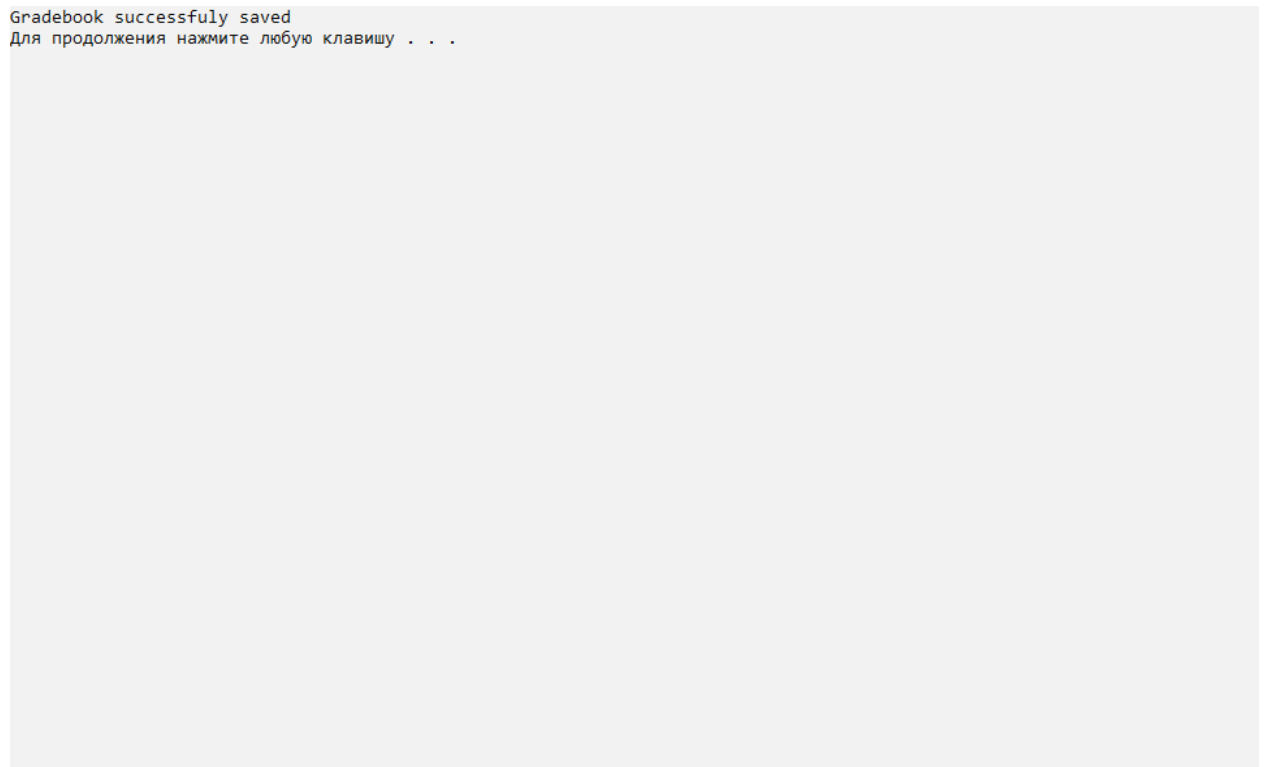
Subject
Subject name: Physics
Hours number: 20

Exam
Date: 10:15 7.6.2021
Mark: 99

-----
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.5 – Show gradebook with data stored in it

Then we have to select "Save gradebook" menu item and if all operations will execute properly, then we will see output message "Gradebook has been successfully saved" like in the figure 3.2.6.



```
Gradebook successfully saved
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.6 – Message on successful save

Then we should exit our program in order to check our loading method. When we run our program again, we could see that our gradebook is empty. To load data, which we saved before we have to select "Load gradebook" menu item. If all operations will execute properly, then we will see output message "Gradebook has been successfully loaded" same as in the figure 3.2.7

Gradebook successfully loaded
 Для продолжения нажмите любую клавишу . . .

Figure 3.2.6 – Message on successful load

Also, we can check our file and we will see our saved data in file "data.txt".
 File data could be seen in the figure 3.2.7.

```

2
Vasiliev
Professor
Hrobovy1
1
Calculus
20
10
15
7
6
120
98

Ivanov
Professor
Hrobovy1
1
Physics
20
10
15
7
5
121
99

```

Figure 3.2.7 – Text file "data.txt"

And finally, we can check our gradebook. We have to select "Show gradebook" menu item and we will see that data, which we saved, is same as our present data.

```

Teacher
Second name: Vasiliev
Post: Professor

Student
Second name: Hrobovyi
Group: 1

Subject
Subject name: Calculus
Hours number: 20

Exam
Date: 10:15 7.7.2020
Mark: 98

-----
Teacher
Second name: Ivanov
Post: Professor

Student
Second name: Hrobovyi
Group: 1

Subject
Subject name: Physics
Hours number: 20

Exam
Date: 10:15 7.6.2021
Mark: 99

-----
Для продолжения нажмите любую клавишу . . .

```

Figure 3.2.8 – Show gradebook after save and load operations

Also, we can check delete data from gradebook. If we select "Delete exam from gradebook" we can delete last exam. If all operations will execute properly, then we will see output message "Exam has been successfully deleted" same as in the figure 3.2.9

```
Exam has been successfully deleted
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.9 – Message on successful delete operation

We can check our gradebook. We have to select "Show gradebook" menu item and we will see that last exam is deleted. We can see it in the figure 3.2.10.

```
Teacher
Second name: Vasiliev
Post: Professor

Student
Second name: Hrobovyi
Group: 1

Subject
Subject name: Calculus
Hours number: 20

Exam
Date: 10:15 7.6.2020
Mark: 98

-----
Для продолжения нажмите любую клавишу . . .
```

Figure 3.2.10 – Message on successful delete operation

We can exit out program by pressing Esc key in the main menu. We can see result in the figure 3.2.11

```
2021-06-07 16:04:50
Esc - exit

Menu:
Show gradebook
Add exam to gradebook
Delete exam from gradebook
Show teacher
Show student
Create teacher
Create student
Save gradebook
Load gradebook

C:\Users\DGB\source\repos\CourseWork\Debug\CourseWork.exe (процесс 16892) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно...
```

Figure 3.2.11 – Closing application program

CONCLUSIONS

In this term work, a program was created, which is an electronic gradebook using the object-oriented approach in writing code. The main tasks of the project were implemented, namely, the code contains elements of object-oriented programming such as classes, objects, a standard set of constructors, virtual methods, getters, setters, overloading operators and destructors for each class, it also has a simple and intuitive interface that allows the user to conveniently use.

REFERENCES

1. Страуструп Б., Программирование: принципы и практика с использованием C++, 2-е изд. : Пер. с англ. – М.: ООО "И.Д. Вильямс", 2016. – 1328 с.
2. Вайсфельд М. Объектно-ориентированное мышление / М. Вайсфельд. – СПб.: Питер, 2014. – 304 с.
3. Кравець П.О. Об'єктно-орієнтоване програмування: навч. посібник / П.О. Кравець. – Львів: Львівська політехніка, 2012. – 624 с.
4. Шилдт, Герберт. C++: руководство для начинающих, 2-е издание. : Пер. с англ./ Г.Шилдт. – М.: Издательский дом "Вильямс", 2005. – 672 с.
5. Дейтел Х., Дейтел П. Как программировать на C++. – М.: БИНОМ, 2008.– 1456 с.

Ministry of Education and Science of Ukraine

SOFTWARE SYSTEM “GRADEBOOK”

Program text

ГЮИК.508100.162 - 01 12 01

sheets 20

File person.h

```
#pragma once
#include <string>
#include <string_view>

class Person
{
public:
    using uint = unsigned int;

    // constructors
    Person();
    Person(std::string_view secondName);
    Person(const Person&) = default;
    Person(Person&&) = default;

    // = operators
    Person& operator=(const Person&) = default;
    Person& operator=(Person&&) = default;

    // virtual destructor
    virtual ~Person();

public:
    // setters & getters for fields
    [[nodiscard]] std::string_view getSecondName() const;
    void setSecondName(std::string_view secondName);

    // pure virtual functions
    virtual void print() const = 0;
    virtual void fill() = 0;
    virtual std::string toString() const = 0;

protected:
    std::string secondName;
};
```

File person.cpp

```
#include "../include/person.h"

Person::Person() :
    secondName{ "Unknown" }
{
}

Person::Person(std::string_view secondName):
    secondName{ secondName }
{
}

Person::~~Person()
{
}

std::string_view Person::getSecondName() const
{
    return secondName;
}
```

```
void Person::setSecondName(std::string_view secondName)
{
    if (secondName.size() > 0)
        this->secondName = secondName;
}
```

File student.h

```
#pragma once
#include <string>

#include "person.h"

class Student : public virtual Person
{
public:
    using uint = unsigned int;

    // constructors
    Student();
    Student(std::string_view secondName, uint group);
    Student(const Student&) = default;
    Student(Student&&) = default;

    // = operators
    Student& operator=(const Student&) = default;
    Student& operator=(Student&&) = default;

    // i/o operators
    friend std::ostream& operator<<(std::ostream& os, const Student& other);
    friend std::istream& operator>>(std::istream& is, Student& other);

public:
    // setters & getters for fields
    [[nodiscard]] uint getGroup() const noexcept;
    void setGroup(uint group);

    // virtual functions
    virtual std::string toString() const override;
    virtual void print() const override;
    virtual void fill() override;

protected:
    std::string secondName;
    uint group;
};
```

File student.cpp

```
#include "../include/student.h"
#include <sstream>
#include <iostream>

Student::Student() :
    secondName{ "Unknown" },
    group{}
{
}

Student::Student(std::string_view secondName, uint group) :
    secondName{ secondName },
```

```

        group{ group }
    {
    }

    unsigned int Student::getGroup() const noexcept
    {
        return group;
    }

    void Student::setGroup(uint group)
    {
        this->group = group;
    }

    std::string Student::toString() const
    {
        std::stringstream ss;
        ss << "secondName: " << secondName << std::endl
            << "group: " << group << std::endl << std::endl;
        return ss.str();
    }

    void Student::print() const
    {
        std::cout << "Second name: " << secondName << std::endl
            << "Group: " << group << std::endl;
    }

    void Student::fill()
    {
        std::cout << "Please enter student's data: " << std::endl;
        std::cout << "Second name: "; std::cin.ignore(32767, '\n'); std::cin >>
secondName;
        std::cout << "Group: "; std::cin >> group;
    }

    std::ostream& operator<<(std::ostream& os, const Student& other)
    {
        os << other.secondName << std::endl
            << other.group << std::endl;
        return os;
    }

    std::istream& operator>>(std::istream& is, Student& other)
    {
        is >> other.secondName >> other.group;
        return is;
    }

```

File teacher.h

```

#pragma once
#include <string>

#include "person.h"

class Teacher : public virtual Person
{
public:
    using uint = unsigned int;

    // constructors
    Teacher();

```

```

    Teacher(std::string_view secondName, std::string_view post);
    Teacher(const Teacher&) = default;
    Teacher(Teacher&&) = default;

    // = operators
    Teacher& operator=(const Teacher&) = default;
    Teacher& operator=(Teacher&&) = default;

    // i/o operators
    friend std::ostream& operator<<(std::ostream& os, const Teacher& other);
    friend std::istream& operator>>(std::istream& is, Teacher& other);

public:
    // setters & getters for fields
    [[nodiscard]] std::string_view getPost() const;
    void setPost(std::string_view post);

    // virtual functions
    virtual std::string toString() const override;
    virtual void print() const override;
    virtual void fill() override;

protected:
    std::string secondName;
    std::string post;
};

```

File teacher.cpp

```

#include "../include/teacher.h"
#include <sstream>
#include <iostream>

Teacher::Teacher() :
    secondName{ "Unknown" },
    post{ "Unknown" }
{
}

Teacher::Teacher(std::string_view secondName, std::string_view post):
    secondName{ secondName },
    post{ post }
{
}

std::string_view Teacher::getPost() const
{
    return post;
}

void Teacher::setPost(std::string_view post)
{
    if (post.size() > 0)
        this->post = post;
}

std::string Teacher::toString() const
{
    std::stringstream ss;
    ss << "secondName: " << secondName << std::endl
        << "post: " << post << std::endl << std::endl;
    return ss.str();
}

```

```

void Teacher::print() const
{
    std::cout << "Second name: " << secondName << std::endl
               << "Post: " << post << std::endl;
}

void Teacher::fill()
{
    std::cout << "Please enter teacher's data: " << std::endl;
    std::cout << "Second name: "; std::cin >> secondName;
    std::cout << "Post: "; std::cin >> post;
}

std::ostream& operator<<(std::ostream& os, const Teacher& other)
{
    os << other.secondName << std::endl
       << other.post << std::endl;
    return os;
}

std::istream& operator>>(std::istream& is, Teacher& other)
{
    is >> other.secondName >> other.post;
    return is;
}

```

File exam.h

```

#pragma once
#include "teacher.h"
#include "student.h"

class Exam:
    public Teacher,
    public Student
{
public:
    using uint = unsigned int;

    Exam();
    Exam(std::string_view teatherSecondName,
         std::string_view post,
         std::string_view studentSecondName,
         uint group,
         std::string_view subjectName,
         uint hoursNumber,
         std::tm date,
         uint mark);

    Exam(const Exam&) = default;
    Exam(Exam&&) = default;
    Exam& operator=(const Exam&) = default;
    Exam& operator=(Exam&&) = default;
    friend std::ostream& operator<<(std::ostream& os, const Exam& other);
    friend std::istream& operator>>(std::istream& is, Exam& other);

public:
    // setters & getters for fields
    [[nodiscard]] std::string_view getSubjectName() const;
    void setSubjectName(std::string_view subjectName);

```

```

[[nodiscard]] uint getHoursNumber() const noexcept;
void setHoursNumber(uint subjectName);

[[nodiscard]] std::tm getDate() const noexcept;
void setDate(std::tm date);

[[nodiscard]] uint getMark() const noexcept;
void setMark(uint mark);

// virtual functions
virtual std::string toString() const override;
virtual void print() const override;
virtual void fill() override;

protected:
    std::string subjectName;
    uint hoursNumber;
    std::tm date;
    uint mark;
};

```

File exam.cpp

```

#include "../include/exam.h"
#include <sstream>
#include <ctime>
#include <iostream>
#include <iomanip>

Exam::Exam() :
    Teacher(),
    Student(),
    subjectName{ "Unknown" },
    hoursNumber{ 1 },
    mark{}
{
    // setting default value for date
    time_t now = time(0);
    gmtime_s(&date, &now);
}

Exam::Exam(std::string_view teacherSecondName, std::string_view post,
    std::string_view studentSecondName, uint group, std::string_view subjectName,
    uint hoursNumber, std::tm date, uint mark):
    Teacher(teacherSecondName, post),
    Student(studentSecondName, group),
    subjectName{ subjectName },
    hoursNumber{ hoursNumber },
    date{ date },
    mark{ mark }
{
}

std::string_view Exam::getSubjectName() const
{
    return subjectName;
}

void Exam::setSubjectName(std::string_view subjectName)
{
    if (subjectName.size() > 0)
        this->subjectName = subjectName;
}

```



```

        else
            throw std::exception{ std::runtime_error{ "Can't set an empty string"} };
    }

    unsigned int Exam::getHoursNumber() const noexcept
    {
        return hoursNumber;
    }

    void Exam::setHoursNumber(uint hoursNumber)
    {
        if (hoursNumber > 0u)
            this->hoursNumber = hoursNumber;
        else
            throw std::exception{ std::runtime_error{ "Can't set 0 hours for a
subject"} };
    }

    std::tm Exam::getDate() const noexcept
    {
        return date;
    }

    void Exam::setDate(std::tm date)
    {
        this->date = date;
    }

    unsigned int Exam::getMark() const noexcept
    {
        return mark;
    }

    void Exam::setMark(uint mark)
    {
        this->mark = mark;
    }

    std::string Exam::toString() const
    {
        std::stringstream ss;
        ss << "teacherSecondName: " << Teacher::secondName << std::endl
            << "post: " << post << std::endl
            << "studentSecondName: " << Student::secondName << std::endl
            << "group: " << Student::group << std::endl
            << "subjectName: " << subjectName << std::endl
            << "hoursNumber: " << hoursNumber << std::endl
            << "date: " << date.tm_hour << ':' << date.tm_min << ' ' << date.tm_mday <<
            << "date: " << date.tm_mon + 1 << ' ' << 1900 + date.tm_year << std::endl
            << "mark: " << mark << std::endl << std::endl;
        return ss.str();
    }

    void Exam::print() const
    {
        std::cout << "Teacher" << std::endl
            << "Second name: " << Teacher::secondName << std::endl
            << "Post: " << Teacher::post << std::endl
            << std::endl

            << "Student" << std::endl
            << "Second name: " << Student::secondName << std::endl
            << "Group: " << Student::group << std::endl
            << std::endl
    }

```

```

        << "Subject" << std::endl
        << "Subject name: " << subjectName << std::endl
        << "Hours number: " << hoursNumber << std::endl
        << std::endl

        << "Exam " << std::endl
        << "Date: " << date.tm_hour << ':' << date.tm_min << ' ' << date.tm_mday <<
        '.' << date.tm_mon + 1 << '.' << 1900 + date.tm_year << std::endl
        << "Mark: " << mark << std::endl
        << std::endl;
    }

void Exam::fill()
{
    std::cout << "Please enter exam data" << std::endl << std::endl;

    Teacher::fill();
    std::cout << std::endl;

    Student::fill();
    std::cout << std::endl;

    std::cout << "Subject data:" << std::endl;
    std::cout << "Subject name: "; std::cin >> subjectName;
    std::cout << "Hours number: "; std::cin >> hoursNumber;

    std::cout << std::endl << "Exam: " << std::endl;
    std::cout << "Date(format: 10:45-9.05.2021): ";
    std::string str;
    std::cin.ignore(32767, '\n');
    std::cin >> str;
    std::stringstream ss(str);
    ss >> std::get_time(&date, "%H:%M-%d.%m.%y");

    if (ss.fail())
        throw std::exception{ std::runtime_error{"Data parse failed. Wrong data."}
};

    std::cout << "Mark: "; std::cin >> mark;
}

std::ostream& operator<<(std::ostream& os, const Exam& other)
{
    os << other.Teacher::secondName << std::endl
        << other.Teacher::post << std::endl
        << other.Student::secondName << std::endl
        << other.Student::group << std::endl
        << other.subjectName << std::endl
        << other.hoursNumber << std::endl
        << other.date.tm_hour << std::endl
        << other.date.tm_min << std::endl
        << other.date.tm_mday << std::endl
        << other.date.tm_mon << std::endl
        << other.date.tm_year << std::endl
        << other.mark << std::endl;
    return os;
}

std::istream& operator>>(std::istream& is, Exam& other)
{
    is >> other.Teacher::secondName
        >> other.Teacher::post
        >> other.Student::secondName

```

```

        >> other.Student::group
        >> other.subjectName
        >> other.hoursNumber
        >> other.date.tm_hour
        >> other.date.tm_min
        >> other.date.tm_mday
        >> other.date.tm_mon
        >> other.date.tm_year
        >> other.mark;
    return is;
}

```

File gradebook.h

```

#pragma once

#include <iterator>
#include "exam.h"

class Gradebook
{
public:
    // constructors
    Gradebook();
    Gradebook(std::initializer_list<Exam> list);
    Gradebook(const Gradebook&);
    Gradebook(Gradebook&&) noexcept;

    // = operators
    Gradebook& operator=(const Gradebook&);
    Gradebook& operator=(Gradebook&&) noexcept;

    // virtual destructor
    virtual ~Gradebook();

public:
    // iterator class
    class Iterator
    {
    public:
        // constructors
        explicit Iterator(Exam* p) : p(p) {}
        Iterator(const Iterator& it) : p(it.p) {}

    public:
        // suffix & postfix ++ operators
        Iterator& operator++() { ++p; return *this; }
        Iterator operator++(int) { Iterator returnValue = *this; ++(*this); return
returnValue; }

        // suffix & postfix -- operators
        Iterator& operator--() { --p; return *this; }
        Iterator operator--(int) { Iterator returnValue = *this; --(*this); return
returnValue; }

        // equality operators
        bool operator==(const Iterator& other) const { return p == other.p; }
        bool operator!=(const Iterator& other) const { return p != other.p; }

        // dereference operator
        Exam& operator*() { return *p; }
    }
}

```

```

        protected:
            // current element
            Exam* p;
        };

public:
    // returns size
    [[nodiscard]] size_t size() const noexcept;

    // adds an element to the container
    void push_back(const Exam&);
    void push_back(Exam&&);

    // deletes an element from the container
    void pop_back();

    // returns iterators referred to beginning and end of the container
    Iterator begin() const noexcept { return Iterator(exams); }
    Iterator end() const noexcept { return Iterator(exams + size_); }

    // saves/loads container to/from file
    void save(std::string_view fileName);
    void load(std::string_view fileName);

    // cleans the container
    void clean();
    // returns true if container is empty, otherwise returns false
    bool empty() const noexcept;

    // prints the container to the console
    void print() const;

protected:
    // pointer to container data
    Exam* exams;
    // container size
    size_t size_;
};

```

File gradebook.cpp

```

#include "../include/gradebook.h"
#include <stdexcept>
#include <fstream>
#include <iostream>

Gradebook::Gradebook() : exams{ nullptr }, size_{} {}

Gradebook::Gradebook(std::initializer_list<Exam> list) :
    size_{ list.size() }
{
    // allocating buffer
    exams = new Exam[size_];

    // copying elements
    size_t i = 0u;
    for (auto it = list.begin(); it != list.end() and i < list.size(); ++it, ++i)
        exams[i] = *it;
}

Gradebook::Gradebook(const Gradebook& other) :
    size_{ other.size_ }

```

```

{
    // allocating buffer
    exams = new Exam[size_];

    // copying elements
    size_t i = 0;
    for (auto it = other.begin(); it != other.end(); ++it, ++i)
        exams[i] = *it;
}

Gradebook::Gradebook(Gradebook&& other) noexcept
{
    // allocating buffer
    exams = new Exam[size_];

    // copying elements
    size_t i = 0;
    for (auto it = other.begin(); it != other.end(); ++it, ++i)
        exams[i] = *it;
}

Gradebook::~Gradebook()
{
    if (exams != nullptr)
        delete[] exams;
}

Gradebook& Gradebook::operator=(const Gradebook& other)
{
    // deleting previous container
    if (exams != nullptr) { delete[] exams; exams = nullptr; }

    // copying elements
    size_ = other.size_;
    exams = new Exam[size_];
    size_t i = 0;
    for (auto it = other.begin(); it != other.end(); ++it, ++i)
        exams[i] = *it;

    return *this;
}

Gradebook& Gradebook::operator=(Gradebook&& other) noexcept
{
    // deleting previous container
    if (exams != nullptr) { delete[] exams; exams = nullptr; }

    // copying elements
    size_ = other.size_;
    exams = new Exam[size_];
    size_t i = 0;
    for (auto it = other.begin(); it != other.end(); ++it, ++i)
        exams[i] = *it;

    return *this;
}

size_t Gradebook::size() const noexcept
{
    return size_;
}

void Gradebook::push_back(const Exam& exam)
{

```

```

    // resizing container
    Exam* newExams = new Exam[size_ + 1];

    // copying elements
    if (!empty()) {
        int i = 0;
        for (auto it = begin(); it != end(); ++it, ++i)
            newExams[i] = *it;
    }
    newExams[size_] = exam;

    // deleting previous container
    if (exams != nullptr) { delete[] exams; exams = nullptr; }
    exams = newExams;
    ++size_;
}

void Gradebook::push_back(Exam&& exam)
{
    // resizing container
    Exam* newExams = new Exam[size_ + 1];

    // copying elements
    if (!empty()) {
        int i = 0;
        for (auto it = begin(); it != end(); ++it, ++i)
            newExams[i] = *it;
    }
    newExams[size_] = exam;

    // deleting previous container
    if (exams != nullptr) { delete[] exams; exams = nullptr; }
    exams = newExams;
    ++size_;
}

void Gradebook::pop_back()
{
    // if container is already empty, throw exeption
    if (!size_)
        throw std::exception{ std::runtime_error{ "Container is already empty" } };

    // resizing container
    --size_;
    Exam* newExams = new Exam[size_];

    // copying elements
    size_t i = 0;
    for (auto it = begin(); i < size_; ++it, ++i)
        newExams[i] = *it;

    // deleting previous container
    if (exams != nullptr) { delete[] exams; exams = nullptr; }

    exams = newExams;
}

bool Gradebook::empty() const noexcept
{
    return !size_;
}

void Gradebook::print() const
{

```

```

        if (empty()) { std::cout << "Gradebook is empty" << std::endl; return; }
        for (auto i = begin(); i != end(); ++i)
        {
            (*i).print();
            std::cout << "-----" <<
std::endl;
        }
    }

void Gradebook::save(std::string_view fileName)
{
    // opening file
    std::ofstream file(fileName.data(), std::ios_base::out);

    // writing data from file
    file << size_ << std::endl;
    for (auto i = begin(); i != end(); ++i)
        file << *i;
    file.close();

    if (file.good())
        std::cout << "Gradebook successfully saved" << std::endl;
    else
        throw std::exception{ std::runtime_error{"Can't save gradebook"} };
}

void Gradebook::load(std::string_view fileName)
{
    // opening file
    std::ifstream file(fileName.data(), std::ios_base::in);

    // reading data from file
    Exam exam;
    size_t size;
    file >> size;
    for (size_t i = 0; !file.eof() and i < size; ++i)
    {
        file >> exam;
        push_back(std::move(exam));
    }
    file.close();

    if (file.good())
        std::cout << "Gradebook successfully loaded" << std::endl;
    else
        throw std::exception{ std::runtime_error{"Can't load gradebook, file is
corrupted"} };
}

void Gradebook::clean()
{
    if (exams != nullptr) { delete[] exams; exams = nullptr; }
    size_ = 0;
}

```

File menu.h

```

#include <Windows.h>
#include <functional>
#include <iostream>
#include <string>

```

```

#include <vector>
#include <conio.h>
#include <chrono>
#include <thread>
#include <mutex>

namespace menu
{
    // menu class
    class Menu
    {
    public:
        using action_t = std::function<void()>;

        // constructors
        Menu(const HANDLE&, const std::string_view& name = "Menu:");
        Menu(const Menu&) = default;
        Menu(Menu&&) = default;

        // = operators
        Menu& operator= (const Menu&) = default;
        Menu& operator= (Menu&&) = default;

        // destructor
        ~Menu();

    public:
        // changes menu name
        void ChangeName(std::string_view);
        // adds an menu item
        void AddItem(std::string_view, action_t);
        // starts up menu
        void GetInput();

    protected:
        // initializes menu
        void Initialize();
        // selects item
        void SelectItem(int position);
        // sets console cursor to the end
        void SetCursorToEnd();
        // shows menu
        void ShowMenu() const;
        // shows time
        void ShowTime() const;
        // shows tooltip
        void ShowTooltip() const;
        // shows items
        void ShowItems() const;
        // shows/hides console cursor
        void ShowCursor(bool);

    protected:
        // struct for items of menu
        struct MenuItem
        {
            // constructor
            MenuItem(std::string_view name_, action_t action_, COORD position_ = COORD{})
                : name{ name_ }, action{ action_ }, position{ position_ }
            {
            }

            // fields
            std::string name;

```



```

        action_t action;
        COORD position;
    };

    // timer struct for menu
    struct Timer
    {
        // constructor
        Timer() : isWorking{ false } {}
        // initializes another thread
        void InitializeThread(Menu& menu);

        // thread flag
        bool isWorking;
        // last timer check
        std::chrono::system_clock::time_point lastCheckedTime;
        // time thread
        std::thread timeThread;
    };

protected:
    // checks time for timer
    void CheckTime(Timer&);
    // menu timer
    Timer timer;
    // mutex for console
    std::mutex lock;

protected:
    // enumeration for keyboard keys
    enum class Keys { ARROW = 224, ARROW_UP = 80, ARROW_DOWN = 72, ENTER = 13, ESCAPE
= 27 };
    // console handles
    HANDLE hConsole;
    // menu position
    int position;
    // menu name
    std::string menuName;
    // menu tooltip
    std::string tooltip;
    // vector of menu items
    std::vector<MenuItem> menuItems;
};
}

```

File menu.cpp

```

#include "../include/menu.h"

namespace menu
{
    Menu::Menu(const HANDLE& hConsole_, const std::string_view& name_) :
        hConsole{ hConsole_ },
        menuName{ name_ },
        position{ 0 },
        tooltip{ "Esc - exit" }
    {
        ShowCursor(false);
    }

    void Menu::AddItem(std::string_view name, action_t action)

```

```

{
    menuItems.push_back(MenuItem(name, action));
}

void Menu::GetInput()
{
    if (menuItems.empty())
        return;

    system("cls");
    Initialize();
    // initial item
    SelectItem(0);

    // init timer thread
    timer.InitializeThread(*this);

    // current key code
    auto key = 0;

    // while not pressed escape button
    while (key != static_cast<int>(Keys::Escape))
    {
        // get pressed key
        key = _getch();
        if (key == static_cast<int>(Keys::Enter))
        {
            { // lock_guard scope
                // locks console mutex
                std::lock_guard<std::mutex> guard(lock);
                system("cls");
                ShowCursor(true);

                // triggers action of an item
                menuItems[position].action();
                system("pause");

                // repaint menu after end of the action
                system("cls");
                ShowCursor(false);
                ShowMenu();
            }
            SelectItem(position);
        }
        if (key != static_cast<int>(Keys::Arrow))
            continue;
        key = _getch();
        const auto selected = position;

        if (key == static_cast<int>(Keys::ArrowUp)) ++position;
        else if (key == static_cast<int>(Keys::ArrowDown)) --position;

        // goes to last item
        if (position < 0)
            position = menuItems.size() - 1;
        // goes to first item
        else if (position > menuItems.size() - 1)
            position = 0;

        // selection
        if (selected != position)
            SelectItem(position);
    }
}

```

```

void Menu::ChangeName(std::string_view name)
{
    menuName = name;
}

void Menu::Initialize()
{
    ShowTime();
    ShowTooltip();
    CONSOLE_SCREEN_BUFFER_INFO screenBuffer{};
    // remember positions of items
    for (auto& item : menuItems)
    {
        GetConsoleScreenBufferInfo(hConsole, &screenBuffer);
        item.position = screenBuffer.dwCursorPosition;
        std::cout << item.name << std::endl;
    }
}

void Menu::SelectItem(int position)
{
    // locks console mutex
    std::lock_guard <std::mutex> guard(lock);
    system("color F0");
    SetConsoleCursorPosition(hConsole, menuItems[position].position);
    // change color
    SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 15));
    std::cout << menuItems[position].name;
    SetConsoleTextAttribute(hConsole, (WORD)((15 << 4) | 0));
    // return color

    SetCursorToEnd();
}

void Menu::SetCursorToEnd()
{
    SetConsoleCursorPosition(hConsole, menuItems[menuItems.size() - 1].position);
    std::cout << std::endl;
}

void Menu::ShowMenu() const
{
    ShowTime();
    ShowTooltip();
    ShowItems();
}

void Menu::ShowTime() const
{
    SYSTEMTIME st;
    GetLocalTime(&st);
    printf("%d-%02d-%02d %02d:%02d:%02d",
        st.wYear,
        st.wMonth,
        st.wDay,
        st.wHour,
        st.wMinute,
        st.wSecond);
    std::cout << std::endl;
}

void Menu::ShowTooltip() const
{

```

```

        std::cout << tooltip << std::endl << std::endl << menuName << std::endl;
    }

    void Menu::ShowItems() const
    {
        for (auto& item : menuItems)
            std::cout << item.name << std::endl;
    }

    void Menu::ShowCursor(bool visibility)
    {
        // making cursor visible/invisible
        CONSOLE_CURSOR_INFO structCursorInfo;
        GetConsoleCursorInfo(hConsole, &structCursorInfo);
        structCursorInfo.bVisible = visibility;
        SetConsoleCursorInfo(hConsole, &structCursorInfo);
    }

    void Menu::Timer::InitializeThread(Menu& menu)
    {
        isWorking = true;
        lastCheckedTime = std::chrono::system_clock::now() + std::chrono::seconds{ 1 };
        // moves CheckTime function to the another thread using lambda expression
        timeThread = std::thread{ [&menu, this]() { menu.CheckTime(*this); } };
    }

    void Menu::CheckTime(Timer& timer)
    {
        COORD timePosition{};
        // check time every 50ms
        while (timer.isWorking)
        {
            if (timer.lastCheckedTime <= std::chrono::system_clock::now())
            {
                // locks console mutex
                std::lock_guard <std::mutex> guard(lock);
                SetConsoleCursorPosition(hConsole, timePosition);
                ShowTime();
                SetCursorToEnd();
            }
            std::this_thread::sleep_for(std::chrono::milliseconds{ 50 });
        }
    }

    Menu::~Menu()
    {
        // close timer thread
        timer.isWorking = false;
        if (timer.timeThread.joinable())
            timer.timeThread.join();
    }
}

```

File main.cpp

```

#include <iostream>

#include "../include/person.h"
#include "../include/student.h"
#include "../include/teacher.h"
#include "../include/exam.h"
#include "../include/gradebook.h"

```

```

#include "../include/menu.h"

constexpr const std::string_view fileName = "data.txt";

int main() {
    using namespace menu;

    // console color
    system("color F0");

    Gradebook gradebook;
    std::unique_ptr<Person> teacher(new Teacher);
    std::unique_ptr<Person> student(new Student);

    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    Menu menu(hConsole);

    menu.AddItem("Show gradebook", [&gradebook] { gradebook.print(); });
    menu.AddItem("Add exam to gradebook", [&gradebook] {
        try {
            Exam exam;
            exam.fill();
            gradebook.push_back(std::move(exam));
        }
        catch (std::exception& expection) {
            std::cout << expection.what() << std::endl;
        }
    });
    menu.AddItem("Delete exam from gradebook", [&gradebook] {
        try {
            gradebook.pop_back();
        }
        catch (std::exception& expection) {
            std::cout << expection.what() << std::endl;
        }
    });

    menu.AddItem("Show teacher", [&teacher] { teacher->print(); });
    menu.AddItem("Show student", [&student] { student->print(); });

    menu.AddItem("Create teacher", [&teacher] {
        try {
            teacher->fill();
        }
        catch (std::exception& expection) {
            std::cout << expection.what() << std::endl;
        }
    });
    menu.AddItem("Create student", [&student] {
        try {
            student->fill();
        }
        catch (std::exception& expection) {
            std::cout << expection.what() << std::endl;
        }
    });

    menu.AddItem("Save gradebook", [&gradebook] {
        try {
            gradebook.save(fileName);
        }
        catch (std::exception& expection) {
            std::cout << expection.what() << std::endl;
        }
    });
    menu.AddItem("Load gradebook", [&gradebook] {

```

```
        try
        {
            gradebook.clean();
            gradebook.load(fileName);
        }
        catch (std::exception& exeption) {
            std::cout << exeption.what() << std::endl;
        }
    });

    menu.GetInput();

    return 0;
}
```