ЗВІТ

Виконав:

Заночкин Є., КІТ-119а

Дата: 27 квітня 2020 р.

Лабораторна робота №6

ПОЛІМОРФІЗМ

Мета роботи. Отримати знання про парадигму ООП – поліморфізм; навчитися застосовувати отримані знання на практиці.

1.Завдання до роботи

Варіант 6.

Загальне завдання: Модернізувати попередню лабораторну роботу шляхом:

- додавання ще одного класу-спадкоємця до базового класу. Поля обрати самостійно;

- базовий клас зробити абстрактним. Додати абстрактні поля;

- розроблені класи-списки поєднуються до одного класу таким чином, щоб він міг працювати як з базовим класом, так і з його спадкоємцями. При цьому серед полів класу-списку повинен бути лише один масив, що містить усі типи класів ієрархії. Оновити методи, що працюють з цим масивом.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: Task.

Класи-спадкоємці: Inheritor, Inheritor2;

Клас, що має в собі масив базового класу та методи для роботи з ним: List.

Клас, що відображає агрегативні відносини з базовим класом: Student.

Клас, що відображає композитивні відносини з базовим класом: Date.

2.2 Опис змінних

int mark – оцінка за роботу (Task).

int countOfDoneExercises – кількість виконаних завдань (Task).

int studentIndex – індекс студента (Task).

string name – ім’я студента (Task).

int rgz – кількість ргз (Task).

Student age – вік студента (Student).

Date date – дата написання (Date).

int rgzForTeacher – кількість ргз для керівника (Inheritor).

string maleFemale – стать студента (Inheritor2).

int listSize – розмір масиву елементів класу List.

Task** studentList – масив елементів класу Task.

List list – об’єкт класу List.

Student age – об’єкт класу Student.

Student* studentAge – масив, що зберігає інформацію про вік кожного студента

List test – об’єкт класу List.

2.3 Опис методів

Task** createList(int, Student*) – створення масиву елементів і заповнення даними (List).

Task** newStudent(Student*, int) – дані стандартних студентів (List).

void printAll(Task**) const – виведення даних елементів у консоль (List).

void printOneStudent(stringstream&) const – виведення даних одного студента у консоль (List).

Task** addStudent(Task*, Task**) – додавання нового елементу в масив (List).

Task** deleteStudent(int, Task**) – видалення елемента з масиву (List).

int getStudentID(int, Task**)const – отримання даних елемента по індексу (List).

int getStudentRGZ(int, Task**)const - отримання даних елемента по кількості ргз (List).

Task** ReadFile(string) – читання даних з файлу (List).

int FileString(string) – кількість рядків у файлі (List).

void WriteFile(string) const – запис даних у файл (List).

Task** enterNewStudent() – введення нового студента з клавіатури (List).

int regexTask(List&) – виведення на екран об’єктів, які в полі string мають 2 слова (List).

static bool sortAsc(const int&, const int&) – функція, що визначає напрям сортування (List).

static bool sortDesc(const int&, const int&) – функція, що визначає напрям сортування (List).

void sort(comp) – сортування масиву (List).

virtual void print() const = 0 – чисто-віртуальний метод виводу на екран (Task).

virtual stringstream getStr() const = 0 – чисто-віртуальний метод заповнення рядку інформацією (Task).

virtual void writeInFile(ofstream&) = 0 – чисто-віртуальний метод запису даних у файл (Task).

void print() const override – метод, що виводить об’єкт на екран, залежно від даних у цьому об’єкті (Inheritor, Inheritor2).

stringstream getStr() const override – метод, що заповнює рядок різними даними, залежно від ситуації (Inheritor, Inheritor2).

void writeInFile(ofstream&) override – метод, що записує дані у файл (Inheritor, Inheritor2).

Task() – конструктор за змовчуванням (Task).

Task(int, string, Student, int, int, int, sint, sint, sint) – конструктор з аргументами (Task).

Task(const Task&) – конструктор копіювання (Task).

Inheritor() – конструктор за змовчуванням (Inheritor).

Inheritor(int, string, Student, int, int, int, sint, sint, sint, int) – конструктор з аргументами (Inheritor).

Inheritor(const Inheritor&) – конструктор копіювання (Inheritor).

Inheritor2() – конструктор за змовчуванням (Inheritor).

Inheritor2(int, string, Student, int, int, int, sint, sint, sint, string) – конструктор з аргументами (Inheritor).

Inheritor2(const Inheritor2&) – конструктор копіювання (Inheritor).

virtual ~Task() – віртуальний деструктор (Task).

~List() – деструктор (List).

~Inheritor() override;

~Inheritor2() override;

2.4 Опис функцій

void Menu() – функція меню.

int generateID() – функція, що надає кожному об’єкту унікальний індекс.

int generateRGZ() – функція, що надає кожному об’єкту унікальну кількість ргз.

Task** TestAddStudent(List&, Student*, Task**) – тест функції додавання об’єкта до масиву об’єктів.

Task** TestDeleteStudent(List&, Task**) – тест функції видалення об’єкта з масиву об’єктів.

void TestGetStudenttID(List&, Task**) – тест функції повернення індексу студента.

void TestReadFile(List&, Task**) – тест функції читання даних з файлу.

void TestSort(List&, Task**) – тест функції сортування масиву.

3. Текст програми

Task.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#define __CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW
#include <string>
```

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <fstream>
#include <regex>
#include <cstddef>
#include "Date.h"
#include "Student.h"
using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::setw;
using std::stringstream;
using std::regex;
using std::ofstream;
using std::ifstream;
typedef bool (comp)(const int&, const int&);
class Task
{
protected:
	int studentIndex;
	string name;
	Student age;
	int mark;
	int countOfDoneExercises;
	int rgz;
	Date date;
public:
	virtual int getMark() const;
	virtual int getCountOfDoneExercises() const;
	virtual int getStudentIndex() const;
	virtual int getRgz() const;
	virtual string getName() const;
	virtual sint getDay() const;
	virtual sint getMonth() const;
	virtual sint getYear() const;
	virtual int getAge() const;
	virtual void print() const = 0;
	virtual stringstream getStr() const = 0;
	virtual void writeInFile(ofstream&) = 0;
	Task();
```

```
        Task(int, string, Student, int, int, int, sint, sint, sint);
        Task(const Task&);
        virtual ~Task();
};
void Menu();
```

List.h

```
#pragma once
#include "Task.h"
#include "Inheritor1.h"
#include "Inheritor2.h"
class List
{
private:
        int listSize;
        Task** studentList;
public:
        int getListSize() const;
        Task** addStudent(Task*, Task**);
        Task* newStudent(Student*, int);
        Task** deleteStudent(int, Task**);
        void printAll(Task**) const;
        void printOneStudent(stringstream&) const;
        Task** createList(int, Student*);
        int getStudentID(int,Task**)const;
        int getStudentRGZ(int,Task**)const;
        Task** ReadFile(string);
        int FileString(string);
        void WriteFile(string) const;
        Task** enterNewStudent();
        int regexTask(List&);
        static bool sortAsc(const int&, const int&);
        static bool sortDesc(const int&, const int&);
        void sort(comp);
        ~List();
};
int generateID();
int generateRGZ();
```

Task.cpp

```
#include "Task.h"
```

```
#include "List.h"
int Task::getMark() const { return mark; }
int Task::getCountOfDoneExercises() const { return countOfDoneExercises; }
int Task::getStudentIndex() const { return studentIndex; }
int Task::getRgz() const { return rgz; }
string Task::getName() const { return name; }
sint Task::getDay() const { return date.getDay(); }
sint Task::getMonth() const { return date.getMonth(); }
sint Task::getYear() const { return date.getYear();; }
int Task::getAge() const { return age.getAge(); }
Task::Task(int student_index, string name, Student age, int mark, int countOfDoneExercises, int rgz, sint day, sint month, sint year) : studentIndex(student_index), name(name), age(age), mark(mark), countOfDoneExercises(countOfDoneExercises), rgz(rgz) ,date(day,month,year) {}
Task::Task() : studentIndex(1), name("Vasya"), mark(5), countOfDoneExercises(5), rgz(5)  {}
Task::Task(const Task& stud) : studentIndex(stud.studentIndex), name(stud.name), age(stud.age), mark(stud.mark), countOfDoneExercises(stud.countOfDoneExercises), rgz(stud.rgz) {}
Task::~Task() {}
```

List.cpp

```
#include "List.h"
#include "Task.h"
#include "Date.h"
int List::getListSize() const { return listSize; }
int generateID()
{
	static int id = 1;
	return id++;
}
int generateRGZ()
{
	static int RGZ = 5;
	return RGZ++;
}
Task** List::addStudent(Task* student, Task** list)
{
	Task** newList = new Task * [listSize+1];
	for (size_t i = 0; i < listSize; i++)
		*(newList + i) = *(list + i);
	newList[listSize++] = student;
	studentList = newList;
	delete list;
	return studentList;
```

```
}
Task** List::createList(int size, Student* age)
{
	studentList = new Task*[size];
	for (size_t i = 0; i<size; i++)
	{
		if (i == 0)
		{
			int id = generateID();
			int rgz = generateRGZ();
			*(studentList + i) = new Inheritor(id, "Vasya", *(age), 2, 5, rgz, 2, 2, 2002, 5);
		}
		else if (i == 1)
		{
			int id = generateID();
			int rgz = generateRGZ();
			*(studentList + i) = new Inheritor2(id, "Ivanova Katya", *(age + 1), 5, 10, rgz, 1,
1, 2000, "Female");
		}
	}
	listSize = size;
	return studentList;
}
Task* List::newStudent(Student* age, int value)
{
	if (value == 1)
	{
		int id = generateID();
		int rgz = generateRGZ();
		Task* Task = new Inheritor(id, "Vasya", *(age), 2, 5, rgz, 2, 2, 2002, 5);
		return Task;
	}
	else
	{
		int id = generateID();
		int rgz = generateRGZ();
		Task* Task = new Inheritor2(id, "Ivanova Katya", *(age + 1), 5, 10, rgz, 1, 1, 2000,
"Female");
		return Task;
	}
}
Task** List::deleteStudent(int value, Task** list)
```

```
{
	if (listSize == 0)
	{
		cout << "Empty list" << endl;
		return NULL;
	}
	Task** newList = new Task * [listSize - 1];
	for (size_t i = 0; i < value; i++)
		*(newList + i) = *(list + i);
	for (size_t i = value, j = value + 1; j < listSize; i++, j++)
		*(newList + i) = *(list + j);
	delete* (studentList + value);
	listSize--;
	studentList = newList;
	delete list;
	return studentList;
}
void List::printOneStudent(stringstream& ss) const
{
	int index, number;
	string name, name2, data;
	int mark, age;
	int exercises;
	int rgz;
	sint day, month, year;
	regex regular("(^[A-ZA-Я]+[\\wA-Za-z,.;:-]* [\\wA-Za-z,.;:-]+)");
	cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) << "Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << setw(12) << "Date" << "Rgz4Teacher/Gender" << endl;
	ss >> index;
	number = getStudentID(index, studentList);
	if (regex_match(studentList[number]->getName(), regular))
	{
		ss >> name;
		ss >> name2;
		ss >> age;
		ss >> mark;
		ss >> exercises;
		ss >> rgz;
		ss >> day;
		ss >> month;
		ss >> year;
```

```
                ss >> data;
                if (name2 == "")
                        name = name + " ";
                else
                        name = name + " " + name2;
        }
        else
        {
                ss >> name;
                ss >> age;
                ss >> mark;
                ss >> exercises;
                ss >> rgz;
                ss >> day;
                ss >> month;
                ss >> year;
                ss >> data;
        }
        cout << std::left;
        cout << setw(6) << index;
        cout << setw(18) << name;
        cout << setw(8) << age;
        cout << setw(13) << mark;
        cout << setw(13) << exercises;
        cout << setw(10) << rgz;
        cout << setw(3) << day << setw(3) << month << setw(12) << year;
        cout << data;
}
void List::printAll(Task** object) const
{
        auto i = 0;
        if (listSize == 0)
        {
                cout << "Empty list" << endl << endl;
                return;
        }
        else if (listSize < i || i < 0)
                cout << "Wrong number" << endl << endl;
        cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) << "Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << setw(12) << "Date" << "Rgz4Teacher/Gender" << endl;
        for (; i < listSize; i++)
```

```
                object[i]->print();
        cout << endl;
    }
    List::~List()
    {
        for (size_t i = 0; i < listSize; i++)
            delete studentList[i];
        delete[] studentList;
    }
    int List::getStudentID(int id,Task** list) const
    {
        for (size_t i = 0; i < listSize; i++)
            if (list[i]->getStudentIndex() == id)
                return i;
        cout << "Wrong ID" << endl;
        return -1;
    }
    int List::getStudentRGZ(int a, Task** list) const
    {
        for (size_t i = 0; i < listSize; i++)
            if (list[i]->getRgz() == a)
                return i;
        cout << "Wrong count of RGZ" << endl;
        return -1;
    }
    Task** List::ReadFile(string filename)
    {
        ifstream fin(filename);
        if (!fin.is_open())
        {
            cout << "Error open file" << endl;
            return NULL;
        }
        int size = List::FileString(filename);
        string line, var;
        regex regularInheritor1("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*
[\\d]* [\\d]*)");
        regex regularInheritor2("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*
[\\d]* [A-Za-z]*)");
        regex regularInheritor1TwoWords("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]*
[\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
```

```
		regex regularInheritor2TwoWords("([\\d]* [A-Z]+[\\wA-ZA-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]*
[\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [A-Za-z]*)");
		size_t i = 0, a = 0, b = 0;
		int index, mark, rgz, exercises, age, rgz4Teach;
		string name, name2, data, gender, surname;
		sint day, month, year;
		for (size_t i = 0; i < listSize; i++)
		{
			delete* (studentList + i);
		}
		delete[] studentList;
		studentList = new Task *[size];
		while (getline(fin, line))
		{
			if (regex_match(line.c_str(), regularInheritor1))
			{
				std::istringstream temp(line);
				temp >> index;
				temp >> surname;
				temp >> age;
				temp >> mark;
				temp >> exercises;
				temp >> rgz;
				temp >> day;
				temp >> month;
				temp >> year;
				temp >> rgz4Teach;
				do {
					b = 0;
					a = name.find("--");
					if (a != -1)
					{
						name.erase(a, 1);
						b = 1;
					}
					a = name.find("  ");
					if (a != -1)
					{
						name.erase(a, 1);
						b = 1;
					}
					a = name.find(",,");
```

```
                        if (a != -1)
                        {
                                name.erase(a, 1);
                                b = 1;
                        }
                        a = name.find("::");
                        if (a != -1)
                        {
                                name.erase(a, 1);
                                b = 1;
                        }
                        a = name.find(";;");
                        if (a != -1)
                        {
                                name.erase(a, 1);
                                b = 1;
                        }
                        a = name.find("_");
                        if (a != -1)
                        {
                                name.erase(a, 1);
                                b = 1;
                        }
                } while (b == 1);
                studentList[i] = new Inheritor(index, surname, age, mark, exercises, rgz, day, month, year, rgz4Teach);
                i++;
        }
        if (regex_match(line.c_str(), regularInheritor2))
        {
                std::istringstream temp(line);
                temp >> index;
                temp >> surname;
                temp >> age;
                temp >> mark;
                temp >> exercises;
                temp >> rgz;
                temp >> day;
                temp >> month;
                temp >> year;
                temp >> gender;
                do {
```

```
                b = 0;
                a = name.find("--");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find("  ");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find(",,");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find("::");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find(";;");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find("_");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
        } while (b == 1);
        studentList[i] = new Inheritor2(index, surname, age, mark, exercises, rgz, day, month, year, gender);
        i++;
}
```

```
if (regex_match(line.c_str(), regularInheritor1TwoWords))
{
	std::istringstream temp(line);
	temp >> index;
	temp >> name;
	temp >> name2;
	temp >> age;
	temp >> mark;
	temp >> exercises;
	temp >> rgz;
	temp >> day;
	temp >> month;
	temp >> year;
	temp >> rgz4Teach;
	if (name2 == "") name = name + " ";
	else(name = name + " " + name2);
	do {
		b = 0;
		a = name.find("--");
		if (a != -1)
		{
			name.erase(a, 1);
			b = 1;
		}
		a = name.find("  ");
		if (a != -1)
		{
			name.erase(a, 1);
			b = 1;
		}
		a = name.find(",,");
		if (a != -1)
		{
			name.erase(a, 1);
			b = 1;
		}
		a = name.find("::");
		if (a != -1)
		{
			name.erase(a, 1);
			b = 1;
		}
```

```
                a = name.find(";;");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find("_");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
        } while (b == 1);
        studentList[i] = new Inheritor(index, name, age, mark, exercises, rgz, day, month, year, rgz4Teach);
        i++;
}
if (regex_match(line.c_str(), regularInheritor2TwoWords))
{
        std::istringstream temp(line);
        temp >> index;
        temp >> name;
        temp >> name2;
        temp >> age;
        temp >> mark;
        temp >> exercises;
        temp >> rgz;
        temp >> day;
        temp >> month;
        temp >> year;
        temp >> gender;
        if (name2 == "") name = name + " ";
        else(name = name + " " + name2);
        do {
                b = 0;
                a = name.find("--");
                if (a != -1)
                {
                        name.erase(a, 1);
                        b = 1;
                }
                a = name.find("  ");
```

```
                                        if (a != -1)
                                        {
                                                name.erase(a, 1);
                                                b = 1;
                                        }
                                        a = name.find(",,");
                                        if (a != -1)
                                        {
                                                name.erase(a, 1);
                                                b = 1;
                                        }
                                        a = name.find("::");
                                        if (a != -1)
                                        {
                                                name.erase(a, 1);
                                                b = 1;
                                        }
                                        a = name.find(";;");
                                        if (a != -1)
                                        {
                                                name.erase(a, 1);
                                                b = 1;
                                        }
                                        a = name.find("_");
                                        if (a != -1)
                                        {
                                                name.erase(a, 1);
                                                b = 1;
                                        }
                                } while (b == 1);
                                studentList[i] = new Inheritor2(index, name, age, mark, exercises, rgz, day, month, year, gender);
                                i++;
                        }
                }
                listSize = size;
                fin.close();
                cout << endl << "Read from file - correct" << endl;
                return studentList;
        }
        int List::FileString(string filename)
        {
```

```
		int c = 0;
		string line;
		regex regularInheritor1("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
		regex regularInheritor2("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [A-Za-z]*)");
		regex regularInheritor1TwoWords("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
		regex regularInheritor2TwoWords("([\\d]* [A-Z]+[\\wA-ZA-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [A-Za-z]*)");
		std::ifstream fin(filename);
		if (!fin.is_open())
		{
			cout << "Error open file";
			return 0;
		}
		while (getline(fin, line))
		{
			if (regex_match(line, regularInheritor1) || regex_match(line, regularInheritor2) || regex_match(line, regularInheritor1TwoWords) || regex_match(line, regularInheritor2TwoWords))
				c++;
			else
				cout << "String is not correct" << endl;
		}
		fin.close();
		return c;
	}
	void List::WriteFile(string filename) const
	{
		std::ofstream fout(filename);
		for (size_t i = 0; i < getListSize(); i++)
			studentList[i]->writeInFile(fout);
		cout << "Write to file - correct" << endl;
		fout.close();
	}
	Task** List::enterNewStudent()
	{
		int index, mark, rgz, exercises, age, rgz4Teach;
		string name, surname, data, gender;
		sint day, month, year;
		regex regularInheritor1("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
```

```
		regex regularInheritor2("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [A-Za-z]*)");
		regex regularInheritor1TwoWords("([\\d]* [A-Z]+[\\wA-Za-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]*)");
		regex regularInheritor2TwoWords("([\\d]* [A-Z]+[\\wA-ZA-z,.;:-]* [A-Z]+[\\wA-Za-z,.;:-]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [\\d]* [A-Za-z]*)");
		cout << "Enter data in line(index,Surname,Name(if it exist),age,mark,exercises,rgz,date(day,month,year),Rgz4Teacher/Gender)" << endl;
		cin.ignore();
		getline(cin, data);
		if (regex_match(data, regularInheritor1))
		{
			std::istringstream temp(data);
			temp >> index;
			temp >> surname;
			temp >> age;
			temp >> mark;
			temp >> exercises;
			temp >> rgz;
			temp >> day;
			temp >> month;
			temp >> year;
			temp >> rgz4Teach;
			Task* Task = new Inheritor(index, surname, age, mark, exercises, rgz, day, month, year, rgz4Teach);
			addStudent(Task, studentList);
		}
		else if (regex_match(data, regularInheritor2))
		{
			std::istringstream temp(data);
			temp >> index;
			temp >> surname;
			temp >> age;
			temp >> mark;
			temp >> exercises;
			temp >> rgz;
			temp >> day;
			temp >> month;
			temp >> year;
			temp >> gender;
			Task* Task = new Inheritor2(index, surname, age, mark, exercises, rgz, day, month, year, gender);
```

```
                addStudent(Task, studentList);
            }
            else if (regex_match(data, regularInheritor1TwoWords))
            {
                std::istringstream temp(data);
                temp >> index;
                temp >> surname;
                temp >> name;
                temp >> age;
                temp >> mark;
                temp >> exercises;
                temp >> rgz;
                temp >> day;
                temp >> month;
                temp >> year;
                temp >> rgz4Teach;
                if (name == "") surname = surname + " ";
                else(surname = surname + " " + name);
                Task* Task = new Inheritor(index, surname, age, mark, exercises, rgz, day, month, year,
rgz4Teach);
                addStudent(Task, studentList);
            }
            else if (regex_match(data, regularInheritor2TwoWords))
            {
                std::istringstream temp(data);
                temp >> index;
                temp >> surname;
                temp >> name;
                temp >> age;
                temp >> mark;
                temp >> exercises;
                temp >> rgz;
                temp >> day;
                temp >> month;
                temp >> year;
                temp >> gender;
                if (name == "") surname = surname + " ";
                else(surname = surname + " " + name);
                Task* Task = new Inheritor2(index, surname, age, mark, exercises, rgz, day, month, year,
gender);
                addStudent(Task, studentList);
            }
```

```
		else
			cout << endl << "You enter wrong data" << endl;
		return studentList;
	}
	int List::regexTask(List& student)
	{
		int value = 0;
		regex regular("(^[A-ZA-Я]+[\\wA-Za-z,.;:-]* [\\wA-Za-z,.;:-]+)");
		cout << std::left << setw(10) << "Index" << setw(13) << "Name" << setw(8) << "Age" << setw(10) << "Mark" << setw(16) << "Exercises" << setw(13) << "RGZ" << setw(12) << "Date" << "Rgz4Teacher/Gender" << endl;
		for (size_t i = 0; i < listSize; i++)
			if (regex_match(student.studentList[i]->getName(), regular))
			{
				student.studentList[i]->print();
				value++;
			}
		cout << endl;
		return value;
	}
	bool List::sortAsc(const int& a, const int& b) { return a > b; }
	bool List::sortDesc(const int& a, const int& b) { return a < b; }
	void List::sort(comp condition)
	{
		Task* temp;
		int pr;
		do
		{
			pr = 0;
			for (size_t i = 0; i < getListSize() - 1; i++)
			{
				if (condition(studentList[i]->getMark(), studentList[i + 1]->getMark()))
				{
					temp = *(studentList + i);
					*(studentList + i) = *(studentList + i + 1);
					*(studentList + i + 1) = temp;
					pr = 1;
				}
			}
		} while (pr == 1);
	}
```

Inheritor1.h

```
#include "Task.h"
#pragma once
class Inheritor : public Task
{
private:
	int rgzForTeacher;
public:
	int getRgzForTeacher() const;
	void setRgzForTeacher(int rgz1);
	void print() const override;
	stringstream getStr() const override;
	void writeInFile(ofstream&) override;
	Inheritor();
	Inheritor(int, string, Student, int, int, int, sint, sint, sint, int);
	Inheritor(const Inheritor& object);
	~Inheritor() override;
};
```

Inheritor2.h

```
#include "Task.h"
#pragma once
class Inheritor2 : public Task
{
private:
	string maleFemale;
public:
	string getMaleFemale() const;
	void setMaleFemale(int);
	void print() const override;
	stringstream getStr() const override;
	void writeInFile(ofstream&) override;
	Inheritor2();
	Inheritor2(int, string, Student, int, int, int, sint, sint, sint, string);
	Inheritor2(const Inheritor2&);
	~Inheritor2() override;
};
```

Inheritor1.cpp

```
#include "Inheritor1.h"
int Inheritor::getRgzForTeacher() const { return rgzForTeacher; }
void Inheritor::setRgzForTeacher(int rgz1) { rgzForTeacher = rgz1; }
```

```
void Inheritor::print() const
{
	cout << setw(6) << studentIndex;
	cout << setw(18) << name;
	cout << setw(8) << age.getAge();
	cout << setw(13) << mark;
	cout << setw(13) << countOfDoneExercises;
	cout << setw(10) << rgz;
	cout << setw(3) << date.getDay() << setw(3) << date.getMonth() << setw(12) << date.getYear();
	cout << rgzForTeacher << endl;
}
stringstream Inheritor::getStr() const
{
	stringstream temp;
	temp << " " << studentIndex << " " << name << " " << age.getAge() << " " << mark << " " <<
countOfDoneExercises
		<< " " << rgz << " " << date.getDay() << " " << date.getMonth()
		<< " " << date.getYear() << " " << rgzForTeacher;
	return temp;
}
void Inheritor::writeInFile(ofstream& el)
{
	el << std::left << setw(6) << studentIndex << setw(18) << name << setw(8) << age.getAge()
		<< setw(10) << mark << setw(13) << countOfDoneExercises << setw(12) << rgz
		<< setw(3) << date.getDay() << setw(3) << date.getMonth()
		<< setw(10) << date.getYear() << rgzForTeacher << endl;
}
Inheritor::Inheritor() : Task(), rgzForTeacher(0) {}
Inheritor::Inheritor(int studentIndex, string name, Student age, int mark, int countOfDoneExercises, int rgz,
sint day, sint month, sint year, int rgzForTeacher) : Task(studentIndex, name, age, mark, countOfDoneExercises,
rgz, day, month, year), rgzForTeacher(rgzForTeacher) {}
Inheritor::Inheritor(const Inheritor& object) : Task(object), rgzForTeacher(object.rgzForTeacher) {}
Inheritor::~Inheritor() {}
```

Inheritor2.cpp

```
#include "Inheritor2.h"
string Inheritor2::getMaleFemale() const { return maleFemale; }
void Inheritor2::setMaleFemale(int rgz1) { maleFemale = rgz1; }
void Inheritor2::print() const
{
	cout << setw(6) << studentIndex;
	cout << setw(18) << name;
```

```
		cout << setw(8) << age.getAge();
		cout << setw(13) << mark;
		cout << setw(13) << countOfDoneExercises;
		cout << setw(10) << rgz;
		cout << setw(3) << date.getDay() << setw(3) << date.getMonth() << setw(12) << date.getYear();
		cout << maleFemale << endl;
	}
	stringstream Inheritor2::getStr() const
	{
		stringstream temp;
		temp << " " << studentIndex << " " << name << " " << age.getAge() << " " << mark << " " <<
countOfDoneExercises
			<< " " << rgz << " " << date.getDay() << " " << date.getMonth()
			<< " " << date.getYear() << " " << maleFemale;
		return temp;
	}
	void Inheritor2::writeInFile(ofstream& el)
	{
		el << std::left << setw(6) << studentIndex << setw(18) << name << setw(8) << age.getAge()
			<< setw(10) << mark << setw(13) << countOfDoneExercises << setw(12) << rgz
			<< setw(3) << date.getDay() << setw(3) << date.getMonth()
			<< setw(10) << date.getYear() << maleFemale << endl;
	}
	Inheritor2::Inheritor2() : Task(), maleFemale("Male") {}
	Inheritor2::Inheritor2(int studentIndex, string name, Student age, int mark, int countOfDoneExercises, int
rgz, sint day, sint month, sint year, string maleFemale) : Task(studentIndex, name, age, mark,
countOfDoneExercises, rgz, day, month, year), maleFemale(maleFemale) {}
	Inheritor2::Inheritor2(const Inheritor2& object) : Task(object), maleFemale(object.maleFemale) {}
	Inheritor2::~Inheritor2() {}
```

Menu.cpp

```
	#include "Task.h"
	#include "List.h"
	#include "Inheritor1.h"
	#include "Inheritor2.h"
	void Menu()
	{
		Task** studentList;
		List list;
		Student age;
		auto c = 0, a = 0, b = 0, value = 0, addNum = 0, count = 0;
		auto count_of_students = 2;
```

```
auto menu_number = 1;
auto delete_number = 0;
string fileName;
stringstream ss;
int del;
Student* studentAge;
studentAge = age.createList(count_of_students);
studentList = list.createList(count_of_students, studentAge);
while (menu_number)
{
	menu_number = 0;
	cout << endl << "Menu:" << endl;
	cout << "1.Add a new student" << endl;
	cout << "2.Add a new student (enter from keyboard)" << endl;
	cout << "3.Delete one student" << endl;
	cout << "4.Show all student" << endl;
	cout << "5.Show student via his index" << endl;
	cout << "6.Show student via his count of RGZ" << endl;
	cout << "7.Read list from file" << endl;
	cout << "8.Write to file" << endl;
	cout << "9.Surname+Name in object" << endl;
	cout << "10.Sort (mark)" << endl;
	cout << "11.End program" << endl;
	cin >> menu_number;
	switch (menu_number)
	{
	case 1:
		cout << "1. Add student with rgz for teacher parameter" << endl;
		cout << "2. Add student with gender parameter" << endl;
		cin >> addNum;
		cout << endl;

		if (addNum == 1)
		{
			Task* newStudent = list.newStudent(studentAge, 1);
			studentList = list.addStudent(newStudent, studentList);
			cout << "Add student - done" << endl;
			break;
		}
		else if (addNum == 2)
		{
			Task* newStudent = list.newStudent(studentAge, 2);
```

```
                studentList = list.addStudent(newStudent, studentList);
                cout << "Add student - done" << endl;
        }
        else
                cout << "Wtong number" << endl;
        break;
case 2:
        studentList = list.enterNewStudent();
        break;
case 3:
        cout << "Enter a index of student who you want to delete:" << endl;
        cin >> delete_number;
        if (delete_number < 1)
        {
                cout << "Wrong student index" << endl;
                break;
        }
        del = list.getStudentID(delete_number, studentList);
        if (del == -1)
                break;
        studentList = list.deleteStudent(del, studentList);
        break;
case 4:
        list.printAll(studentList);
        break;
case 5:
        cout << "Enter a index of student:";
        cin >> c;
        b = list.getStudentID(c, studentList);
        if (b == -1)
                break;
        ss = studentList[b]->getStr();
        list.printOneStudent(ss);
        break;
case 6:
        cout << "Enter a count of RGZ:";
        cin >> a;
        b = list.getStudentRGZ(a, studentList);
        if (b == -1)
                break;
        ss = studentList[b]->getStr();
        list.printOneStudent(ss);
```

```
                break;
            case 7:
                cout << "Enter file name:";
                cin >> fileName;
                studentList = list.ReadFile(fileName);
                break;
            case 8:
                cout << "Enter file name:";
                cin >> fileName;
                list.WriteFile(fileName);
                break;
            case 9:
                count = list.regexTask(list);
                if (count == 0)
                    cout << "There is no students with Name and Surname" << endl;
                break;
            case 10:
                cout << "1) Increasing" << endl;
                cout << "2) Decreasing" << endl;
                cin >> value;
                cout << endl;
                if (value == 1)
                    list.sort(list.sortAsc);
                else if (value == 2)
                    list.sort(list.sortDesc);
                else cout << "Wrong number." << endl;
                break;
            case 11:
                menu_number = 0;
                break;
            default:
                cout << "You have chosen the wrong number of the menu";
                break;
            }
        }
        age.deleteList();
        return;
}
```

main.cpp

```
#include "Task.h"
int main()
```

```
{
Menu();
if (_CrtDumpMemoryLeaks())
cout << endl << "WARNING! Memory leak" << endl;
else
cout << endl << "There is no memory leak" << endl;
return 0;
}
```

Student.h

```
#pragma once
#include <iostream>
#include <string>
using std::string;
class Student {
private:
	int age;
	int listSize;
	Student* list;
public:
	int getAge()const;
	void setAge(const int);
	void deleteList();
	Student* createList(int size);
	Student students(int value);
	Student();
	Student(int);
	Student(const Student& other);
	~Student();
};
```

Date.h

```
#pragma once
typedef short sint;
class Date
{
private:
sint day;
sint month;
sint year;
public:
```

```
sint getDay() const;
sint getMonth() const;
sint getYear() const;
Date();
Date(sint, sint, sint);
Date(const Date& date);
~Date();
};
```

Student.cpp

```
#include "Student.h"
int Student::getAge() const { return age; }
void Student::setAge(const int age1) { age = age1; }
Student* Student::createList(int size)
{
	listSize = size;
	list = new Student[size];
	for (size_t i = 0; i < size; i++)
		list[i] = students(i);
	return list;
}
Student Student::students(int value)
{
	if (value == 0)
	{
		Student age1 = 10;
		return age1;
	}
	else if (value == 1)
	{
		Student age2 = 15;
		return age2;
	}
}
void Student::deleteList()
{
	delete[] list;
}
Student::Student() : age(0) {}
Student::Student(int age) : age(age) {}
Student::Student(const Student& student) : age(student.age) {}
Student::~Student() {}
```

Date.cpp

```
#include "Date.h"
sint Date::getDay() const { return day; }
sint Date::getMonth() const { return month; }
sint Date::getYear() const { return year; }
Date::Date() : day(2), month(2), year(2002) {}
Date::Date(sint day, sint month, sint year) : day(day), month(month), year(year) {}
Date::Date(const Date& date) : day(date.day), month(date.month), year(date.year) {}
Date::~Date() {}
```

Test.cpp

```
#include "List.h"
#include "Task.h"
Task** TestAddStudent(List&, Student*, Task**);
Task** TestDeleteStudent(List&, Task**);
void TestGetStudentID(List&, Task**);
void TestReadFile(List&, Task**);
void TestSort(List&, Task**);
int main()
{
	{
		List test;
		Student age;
		Student* studentAge;
		Task** studentList;
		studentAge = age.createList(2);
		studentList = test.createList(2, studentAge);
		studentList = TestAddStudent(test, studentAge, studentList);
		studentList = TestDeleteStudent(test, studentList);
		TestGetStudentID(test, studentList);
		TestReadFile(test, studentList);
		TestSort(test, studentList);
		age.deleteList();
	}
	if (_CrtDumpMemoryLeaks())
		cout << endl << "WARNING! Memory leak" << endl;
	else
		cout << endl << "There is no memory leak" << endl;
	return 0;
}
Task** TestAddStudent(List& test, Student* age, Task** stud)
```

```
{
	int value = test.getListSize();
	Task* newStudent = test.newStudent(age, 1);
	stud = test.addStudent(newStudent, stud);
	if (test.getListSize() > value)
		cout << endl << "Test: Add_student - successful" << endl;
	else
		cout << endl << "Test: Add_student - unsuccessful" << endl;
	return stud;
}
Task** TestDeleteStudent(List& test, Task** stud)
{
	int size = test.getListSize();
	stud = test.deleteStudent(2, stud);
	int newSize = test.getListSize();
	if (size > newSize)
		cout << endl << "Test: Delete_student - successful" << endl;
	else
		cout << endl << "Test: Delete_student - unsuccessful" << endl;
	return stud;
}
void TestGetStudentID(List& test, Task** stud)
{
	int num = test.getStudentID(2, stud);
	if (num == 1)
		cout << endl << "Test: GetStudentId - successful" << endl;
	else
		cout << endl << "Test: GetStudentId - unsuccessful" << endl;
}
void TestReadFile(List& test, Task** stud)
{
	int expected = 4;
	int real = test.FileString("Text.txt");
	if (expected == real)
		cout << endl << "Test: ReadFile - successful" << endl;
	else
		cout << endl << "Test: ReadFile - unsuccessful" << endl;
}
void TestSort(List& test, Task** stud)
{
	int beforeSorting = stud[0]->getMark();
	test.sort(test.sortDesc);
```

```
    int afterSorting = stud[0]->getMark();
    int expected = 5;

    if (beforeSorting != afterSorting && afterSorting == expected)
        cout << endl << "Test: Sort - successful" << endl;
    else
        cout << endl << "Test: Sort - unsuccessful" << endl;

}
```

4. Результати роботи програми

```
Index     Name          Age     Mark      Exercises       RGZ          Date        Rgz4Teacher/Gender
1     Vasya             10      2           5             5         2  2  2002        5
2     Ivanova Katya     15      5           10            6         1  1  2000        Female


Menu:
1.Add a new student
2.Add a new student (enter from keyboard)
3.Delete one student
4.Show all student
5.Show student via his index
6.Show student via his count of RGZ
7.Read list from file
8.Write to file
9.Surname+Name in object
10.Sort (mark)
11.End program
1
1. Add student with rgz for teacher parameter
2. Add student with gender parameter
1

Add student - done

Menu:
1.Add a new student
2.Add a new student (enter from keyboard)
3.Delete one student
4.Show all student
5.Show student via his index
6.Show student via his count of RGZ
7.Read list from file
8.Write to file
9.Surname+Name in object
10.Sort (mark)
11.End program
4
Index     Name          Age     Mark      Exercises       RGZ          Date        Rgz4Teacher/Gender
1     Vasya             10      2           5             5         2  2  2002        5
2     Ivanova Katya     15      5           10            6         1  1  2000        Female
3     Vasya             10      2           5             7         2  2  2002        5
```

Рисунок 1 – Робота класів-спадкоємців

5. Висновки

Під час виконання лабораторної роботи було додано другий клас-спадкоємець, зроблено базовий клас абстрактним, додано абстрактні поля до базового класу, класи-списки поєднані до одного класу таким чином, що він може працювати як з базовим класом, так і з класами-спадкоємцями, зроблено єдиний масив, що може містити усі типи даних та оновлено методи, що працюють з цим масивом.

Програма протестована, виконується без помилок, витоків пам’яті немає.