МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет

«Харківський політехнічний інститут»

Факультет Комп’ютерних наук та програмної інженерії

Кафедра Інформатики та інтелектуальної власності

ЗВІТ

До лабораторной роботи №6 з дисципліни

«Об’єкто-орієнтоване програмування»

Студент Бородай Д.А

Викладач Івашко А.В.

Харків 2022

**Лабораторна робота №6**

**Тема: «Асоціація, композиція та агрегація в ООП»**

**Мета роботи:** навчитись розрізнювати, створювати та використовувати відносини композицій та агрегацій між класами та групами класів.

**Завдання:**

1. Для класу лабораторної роботи №1 згідно до свого варіанту створити головний чи підпорядкований клас із яким організувати відносини агрегації. Продемонструвати роботу відносин, створивши декілька об’єктів відповідних класів та позбавляючи чи додаючи властивості до цих об’єктів.

2. Для класу лабораторної роботи №1 згідно до свого варіанту створити головний чи підпорядкований клас із яким організувати відносини композиції. Продемонструвати роботу відносин, створивши декілька об’єктів відповідних класів та позбавляючи чи додаючи властивості до цих об’єктів

Зміст

[1 Аналіз до предметної області 4](#_Toc119665362)

[2 Виконання роботи 5](#_Toc119665363)

[3 Довідка по роботі з програмою для користувача. 6](#_Toc119665364)

[Висновок 9](#_Toc119665365)

## 1 Аналіз до предметної області

В об’єктно-орієнтованому загальному проектуванні програмного забезпечення зв’язок між функціональними можливостями одного об’єкта та іншим називається асоціацією. Зауважте, що асоціація між двома об’єктами – це не те ж саме, що спадкування між двома класами. Асоціація означає, що один об’єкт використовує інший об’єкт або функцію/метод у цьому іншому об’єкті. Іншими словами, зв'язок визначається як зв'язок між об'єктами, коли один об'єкт має одне або більше посилань на інші об'єкти.

Існує два типи асоціацій між об’єктами: композиція та агрегація.

Композиція — це форма асоціації, яка виникає, коли життя об’єкта тісно пов’язане з життям іншого об’єкта. Коли головний об’єкт вмирає (тобто видаляється), усі об’єкти, пов’язані з цим об’єктом, також вмирають. Це означає, що об’єкт, на який посилається, міститься виключно об’єктом, на який посилається.

Агрегація є іншою формою асоціації, подібною до композиції. При агрегуванні об’єкт-контейнер знову має кілька посилань на інші об’єкти. Але агрегація вільніша, ніж композиція. Життєві цикли об’єктів не пов’язані один з од-ним у сукупності. Таким чином, об’єкт, на який посилається, може бути знищений до/після об’єкта, на який посилається. За допомогою агрегації на об’єкт завжди посилатимуться інші об’єкти.

## 2 Виконання роботи

Згідно поданих інструкцій, рисунок 2.1, було створено відповідні класи Deparmnet та GradeSub, код котрих описано нижче.

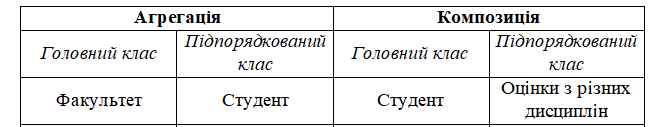


Рисунок 2.1 – Проектування класів

class Department

{

string DepName;

bool fullDay;

string dataDepNames[5] = { "Computer Science", "Computer Modeling", "Integrated Tech. ", "Geom. Modeling", "SGC" };

public:

Department() { DepName = dataDepNames[rand() % 5]; fullDay = rand() % 2; }

string getterDepName() { return DepName; }

bool getterfullDay() { return fullDay; }

};

class GradeSub {

short int grade;

string subject;

public:

void setterGrade(const short int& grade) noexcept { this->grade = grade; } ;

void setterSubject(const string& subject) { this->subject = subject; };

short int getterGrade() noexcept { return grade; };

string getterSubject() { return subject; };

};

Повний код програми наведено у додатку А.

## 3 Довідка по роботі з програмою для користувача.

Головне меню зустрічає користувача п’ятьма пунктами. Перший пункт – це робота вже з існуючим списком, якщо списку не існує, то користувачу програма пропонує його створити. Другий – збереження цього списку на локальному диску, новоствореного чи вже існуючого. Третій пункт – це завантаження до програми вже існуючого списку. Четвертий – це видалення з пам’яті програми списку, з яким користувач працював. Та останній – вихід з програми. Важливо те, що до цього меню можна завжди повернутися, це дає можливість після кожної операції самостійно зберігати список, не втрачаючи данні, або почати роботу вже з новим. Меню наглядно продемонстровано на рисунку 3.1.

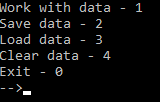


Рисунок 2.1 – Головне меню

У меню по роботі зі списком наведено сім пунктів, перші три з яких відносяться до завдання, що було поставлено раніше. Що продемонстровано на рисунку 2.2.

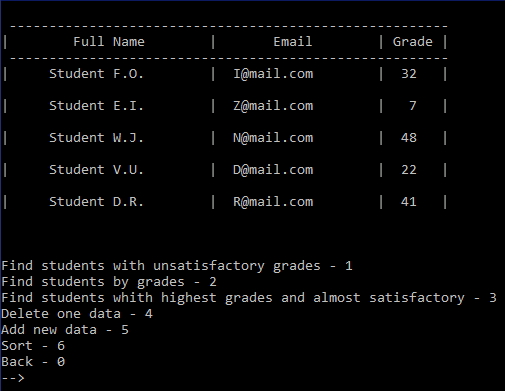


Рисунок 3.2 – Робота зі списком

Пункт четвертий пропонує користувачу видалити одну комірку даних студента у заданому місці, що наведено на рисунку 2.3 та 2.4.



Рисунок 3.3 – Задання номеру комірки для видалення



Рисунок 3.4 – Результат видалення третьої комірки

П’ятий пункт дозволяє навпаки додати після заданої комірки даних, результат роботи наведено на рисунку 2.5 та 2.6.



Рисунок 3.5 – Задання комірки, після якої буде додано нові данні



Рисунок 3.6 – Результат додавання нових даних після другої комірки

У шостому пункті користувач може відсортувати від найбільшого до найменшого оцінки або ФІО, задля зручного особистого пошуку або користування. Роботу пункту продемонстровано на рисунку 2.7.



Рисунок 2.7 – Результат сортування за оцінками

Повертаючись до трьох перших пунктів, у першому – користувач може за допомогою програми віднайти студентів з незадовільними оцінками. У другому пункті – користувач може знайти студентів, які маю не нижчу оцінку, задану ним. Третій пункт дозволяє віднайти студентів з оцінкою вище 89 балів та студентів які мають можливість вийти на перездачу та/або повторний екзамен. Останній пункт дозволяє повернутися до головного меню.

## **Висновок**

Різниця між композицією та агрегацією полягає в тому, що в разі композиції ціле явно контролює час життя своєї складової частини (частина не існує без цілого), а в разі агрегації ціле хоч і містить свою складову частину, час їхнього життя не пов'язаний (наприклад, складова частина передається через параметри конструктора).

**Додаток А**

Header.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <iomanip>

#include <ctime>

#define MAXSIZE 300

using namespace std;

class Department

{

string DepName;

bool fullDay;

string dataDepNames[5] = { "Computer Science", "Computer Modeling", "Integrated Tech. ", "Geom. Modeling", "SGC" };

public:

Department() { DepName = dataDepNames[rand() % 5]; fullDay = rand() % 2; }

string getterDepName() { return DepName; }

bool getterfullDay() { return fullDay; }

};

class Student {

class GradeSub {

short int grade;

string subject;

public:

void setterGrade(const short int& grade) noexcept { this->grade = grade; } ;

void setterSubject(const string& subject) { this->subject = subject; };

short int getterGrade() noexcept { return grade; };

string getterSubject() { return subject; };

};

string FullName;

string address;

GradeSub grade;

Department departament;

public:

Student\* next, \* prev;

//Student();

// setter

void setterAll( string FullName, string address, short int grade, string subject);

//getters

string getterFullname(); string getterAddress(); short int getterGrade(); string getterSubject();

void show();

//~Student(){};

friend void Sort(Student\*\* head, short int set);

};

class Main

{

private:

int size;

public:

Main() : size(0) {};

int GetSize() { return size; };

void CreateList(short int count, Student\*\* head, Student\* tail);

void PrintList(Student\* head);

void Insert(Student\*\* head, Student\*\* tail, int pos);

void DeleteInList(Student\*\* Dhead, Student\*\* Dtail, int pos);

bool DcheckNULL(Student\* head);

void FreeList(Student\*\* head);

void table\_header();

// main tasks

void unsatisfactory(Student\*\* head);

void byGrade(Student\*\* head, short int setted);

void hightesGradeandHalf(Student\*\* head);

// working method

void main();

~Main() {};

};

**Додаток Б**

Source.cpp

#include "Header.h"

string dataNames[32] = { "Ethan", "Josh", "Sam", "Tom", "Bruse", "Michael", "Diego", "Adrian", "Brian", "Carl", "Caleb", "Charles", "Daniel", "Derek", "Elliot", "Kuel",

"Sophia", "Amelia", "Ella", "Chloe", "Penelope", "Emma", "Ava", "Isabella", "Elizabath", "Ginna", "Layla", "Zoey", "Olivia", "Sarah", "Rachel", "Emily" };

string dataSecondNames[50] = { "Smith", "Johnson", "Williams", "Jones", "Brown","Davis","Miller", "Wilson","Moore","Taylor", "Anderson","Thomas","Jackson", "White","Harris","Martin", "Thompson","Garcia","Martinez", "Robinson","Clark","Rodriguez", "Lewis","Lee","Walker", "Hall","Allen","Young", "Hernandez","King","Wright", "Lopez","Hill","Scott", "Green","Adams","Baker",

"Gonzalez", "Nelson","Carter","Mitchell", "Perez","Roberts","Turner", "Phillips","Campbell","Parker", "Evans","Edwards","Collins" };

string dataSubjects[11] = { "Arts", "Bilogy", "History", "Music", "Math", "Chemistry", "Geography", "PE", "Economics", "English", "Polska" };

// main class

void Main::main()

{

short int count, button, checker;

Student\* head = NULL, \* tail = NULL;

while (true) {

cout << "Work with data - 1 \nClear data - 2 \nExit - 0 \n-->"; cin >> button;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

continue;

}

switch (button)

{

case 1:

while (button != 0) {

system("cls");

if (head == NULL) {

cout << "How long you list could be? \n---> "; cin >> count;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

continue;

}

CreateList(count, &head, tail);

system("cls");

}

table\_header();

PrintList(head);

cout << "Find students with unsatisfactory grades - 1\n" <<

"Find students by grades - 2\n" <<

"Find students whith highest grades and almost satisfactory - 3\n" <<

"Delete one data - 4\n" <<

"Add new data - 5\n" <<

"Sort - 6\n" <<

"Back - 0\n" <<

"-->";

cin >> button;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

continue;

system("pause");

}

switch (button)

{

case 1:

system("cls");

unsatisfactory(&head);

system("pause");

break;

case 2:

system("cls");

cout << "Input grade limit \n-->"; cin >> checker;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

system("pause");

continue;

}

byGrade(&head, checker);

system("pause");

break;

case 3:

system("cls");

hightesGradeandHalf(&head);

system("pause");

break;

case 4:

system("cls");

cout << "Input number data \n-->"; cin >> checker;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

system("pause");

continue;

}

DeleteInList(&head, &tail, checker);

break;

case 5:

system("cls");

cout << "Input number data \n-->"; cin >> checker;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

system("pause");

continue;

}

Insert(&head, &tail, checker);

break;

case 6:

system("cls");

cout << "Sort by name - 1 \nSort by grades - 2 \nBack - 0 \n-->"; cin >> button;

if (cin.fail()) {

system("cls");

cin.clear();

cin.ignore(32767, '\n');

cout << " \t\t\tIncorrect input! ERROR! " << endl;

system("pause");

continue;

}

switch (button)

{

case 1:

Sort(&head, button);

table\_header();

PrintList(head);

system("pause");

break;

case 2:

Sort(&head, button);

table\_header();

PrintList(head);

system("pause");

break;

case 0:

system("cls");

break;

default:

break;

}

break;

case 0:

system("cls");

break;

default:

break;

}

}

break;

case 2:

system("cls");

if (DcheckNULL(head) != 0)

FreeList(&head);

break;

case 0:

system("cls");

FreeList(&head);

exit(0);

break;

default:

break;

}

}

}

//Student::Student()

//{

// this->FullName = "User invalid";

// this->address = "None";

// this->grade = -1;

//

//}

//set methods

void Student::setterAll(string FullName, string address, short int grade, string subject)

{

this->FullName = FullName;

this->address = address;

this->grade.setterGrade(grade);

this->grade.setterSubject(subject);

}

// get methods

string Student::getterFullname() { return this->FullName; }

string Student::getterAddress() { return this->address; }

short int Student::getterGrade() { return grade.getterGrade(); }

string Student::getterSubject(){ return grade.getterSubject();}

// show method

void Student::show()

{

cout << "|" << setw(22) << this->FullName << setw(4) << "|" << setw(32) << this->address << setw(4) << "|" << setw(4) << this->grade.getterGrade() << setw(4) << "|" << setw(12) << this->grade.getterSubject() << setw(4) << "|" << setw(18) << this->departament.getterDepName() << setw(4) << "|" << setw(6) << boolalpha << this->departament.getterfullDay() << setw(6) << "|";

cout << "\n -----------------------------------------------------------------------------------------------------------------------\n";

}

///////////////////////////////////////////////////////////////////////

///////////////////// Main tasks ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

void Main::unsatisfactory(Student\*\* head)

{

int count = 0;

Student\* current = \*head;

Student\* unsatisfactoryGrade = new Student[MAXSIZE];

if (DcheckNULL((\*head)) == 0) { return; }

else {

while (current) {

if (current->getterGrade() < 60) {

unsatisfactoryGrade[count].setterAll(current->getterFullname(), current->getterAddress(), current->getterGrade(), current->getterSubject());

count++;

}

current = current->next;

}

}

if (count != 0) {

cout << "\t Students with unsatisfactory grades: " << count << endl;

table\_header();

for (size\_t i = 0; i < count; i++) {

unsatisfactoryGrade[i].show();

}

}

else

cout << "\tStudents haven`t been found!\n";

delete[] unsatisfactoryGrade; unsatisfactoryGrade = NULL;

}

void Main::byGrade(Student\*\* head, short int setted)

{

int count = 0;

Student\* current = \*head;

Student\* byGrade = new Student[MAXSIZE];

if (DcheckNULL((\*head)) == 0) { return; }

else {

while (current) {

if (current->getterGrade() > setted) {

byGrade[count].setterAll(current->getterFullname(), current->getterAddress(), current->getterGrade(), current->getterSubject());

count++;

}

current = current->next;

}

}

if (count != 0) {

cout << "\t Students with grades higher " << setted << ": " << count << endl;

table\_header();

for (size\_t i = 0; i < count; i++) {

byGrade[i].show();

}

}

else

cout << "\tStudents haven`t been found!\n";

delete[] byGrade; byGrade = NULL;

}

void Main::hightesGradeandHalf(Student\*\* head)

{

int count = 0;

Student\* current = \*head;

Student\* byGrade = new Student[MAXSIZE];

if (DcheckNULL((\*head)) == 0) { return; }

else {

while (current) {

if (current->getterGrade() > 89 or (current->getterGrade() > 34 and current->getterGrade() < 60)) {

byGrade[count].setterAll(current->getterFullname(), current->getterAddress(), current->getterGrade(), current->getterSubject());

count++;

}

current = current->next;

}

}

if (count != 0) {

cout << "\tStudents " << ": " << count << endl;

table\_header();

for (size\_t i = 0; i < count; i++) {

byGrade[i].show();

}

if (count != 1) {

short int choose = -1;

Student\* chosen = new Student[count];

short int flag = 0;

short index = 0;

current = \*head;

while (choose != 0 and flag != count) {

cout << "Choose the studenet by number (0-stop) --> "; cin >> choose;

if (choose > count) {

cout << "\tFalse input, try again!\n";

continue;

}

if (choose != 0) { flag++; };

for (size\_t i = 0; i < choose; i++) {

if (i == choose - 1) {

chosen[index] = byGrade[choose - 1];

index++;

}

current = current->next;

}

}

if (flag != 0) {

table\_header();

for (size\_t i = 0; i < flag; i++) {

chosen[i].show();

}

}

else

cout << "\tStudents haven`t been chosen!\n";

delete[] chosen; chosen = NULL;

}

else {

cout << "You shouldn`t to choose the one student!\n";

}

}

else

cout << "\tStudents haven`t been found!\n";

delete[] byGrade; byGrade = NULL;

}

void Main::table\_header()

{

cout << "\n -----------------------------------------------------------------------------------------------------------------------\n";

cout << "|" << setw(17) << "Full Name" << setw(9) << "|" << setw(23) << "Email" << setw(13) << "|" << setw(6) << "Grade" << setw(2) << "|" << setw(12) << "Subject" << setw(4) << "|" << setw(18) << "Department" << setw(4) << "|" << setw(6) << "FTS" << setw(6) << "|";

cout << "\n -----------------------------------------------------------------------------------------------------------------------\n";

}

///////////////////////////////////////////////////////////////////////

///////////////////// Doubly Node ///////////////////////////////////

/////////////////////////////////////////////////////////////////////

void Main::CreateList(short int count, Student\*\* head, Student\* tail)

{

string FullName, buffer, address, subject; short int grade;

if (count > 0) {

(\*head) = new Student;

FullName = dataNames[rand() % 32];

address += FullName;

FullName += " ";

buffer = dataSecondNames[rand() % 32];

FullName += buffer;

address += buffer += "@gmail.com";

grade = rand() % 101;

subject = dataSubjects[rand() % 11];

(\*head)->setterAll(FullName, address, grade, subject);

(\*head)->prev = tail;

(\*head)->next = NULL;

CreateList(count - 1, &((\*head)->next), (\*head));

}

else (\*head) = NULL;

size = count;

}

void Main::PrintList(Student\* head) {

if (head != NULL) {

head->show();

PrintList(head->next);

}

else cout << endl << endl;

}

void Main::Insert(Student\*\* head, Student\*\* tail, int pos) {

Student\* temp = new Student;

if (pos == 0) {

temp->next = \*head;

(\*head)->prev = temp;

\*head = temp;

temp->prev = NULL;

}

else {

Student\* current = \*head;

for (size\_t i = 0; i < pos - 1; i++)

current = current->next;

if (current->next == NULL) {

current->next = temp;

temp->prev = \*tail;

temp->next = NULL;

\*tail = temp;

}

else {

temp->next = current->next;

current->next = temp;

temp->next->prev = temp;

temp->prev = current;

}

}

string FullName, buffer, address, subject; short int grade;

FullName = dataNames[rand() % 32];

address += FullName;

FullName += " ";

buffer = dataSecondNames[rand() % 32];

FullName += buffer;

address += buffer += "@gmail.com";

grade = rand() % 101;

subject = dataSubjects[rand() % 11];

temp->setterAll(FullName, address, grade, subject);

}

void Main::DeleteInList(Student\*\* head, Student\*\* tail, int pos) {

Student\* current = \*head;

if (pos == 0) {

\*head = (\*head)->next;

(\*head)->prev = NULL;

delete current;

}

else {

for (size\_t i = 0; i < pos - 1; i++)

current = current->next;

Student\* temp = current->next;

if (temp->next == NULL) {

current->next = NULL;

\*tail = current;

}

else {

current->next = temp->next;

temp->next->prev = current;

}

delete temp;

}

}

bool Main::DcheckNULL(Student\* head) { // cheking on empty

if (head == NULL) {

cout << "List is empty." << endl;

return 0;

}

return 1;

}

void Main::FreeList(Student\*\* head) { // clean list

Student\* current = \*head;

while (current) {

\*head = (\*head)->next;

free(current);

current = \*head;

}

}

void Sort(Student\*\* head, short int set) {

Student\* left = \*head;

Student\* right = (\*head)->next;

Student\* temp = new Student;

// preset switch

switch (set) {

case 1: // sort by name

while (left->next) {

while (right) {

if ((left->FullName.compare(right->FullName) > 0)) {

temp->FullName = left->FullName;

left->FullName = right->FullName;

right->FullName = temp->FullName;

}

right = right->next;

}

left = left->next;

right = left->next;

}

break;

case 2: // sort by grades

while (left->next) {

while (right) {

if (right->grade.getterGrade() > left->grade.getterGrade()) {

temp->grade.setterGrade(right->grade.getterGrade());

right->grade.setterGrade(left->grade.getterGrade());

left->grade = temp->grade;

}

right = right->next;

}

left = left->next;

right = left->next;

}

break;

default:

//system("cls");

break;

}

}