СТАТИЧНІ МЕТОДИ, ПЕРЕВАНТАЖЕННЯ ОПЕРАТОРІВ ТА МЕТОДІВ

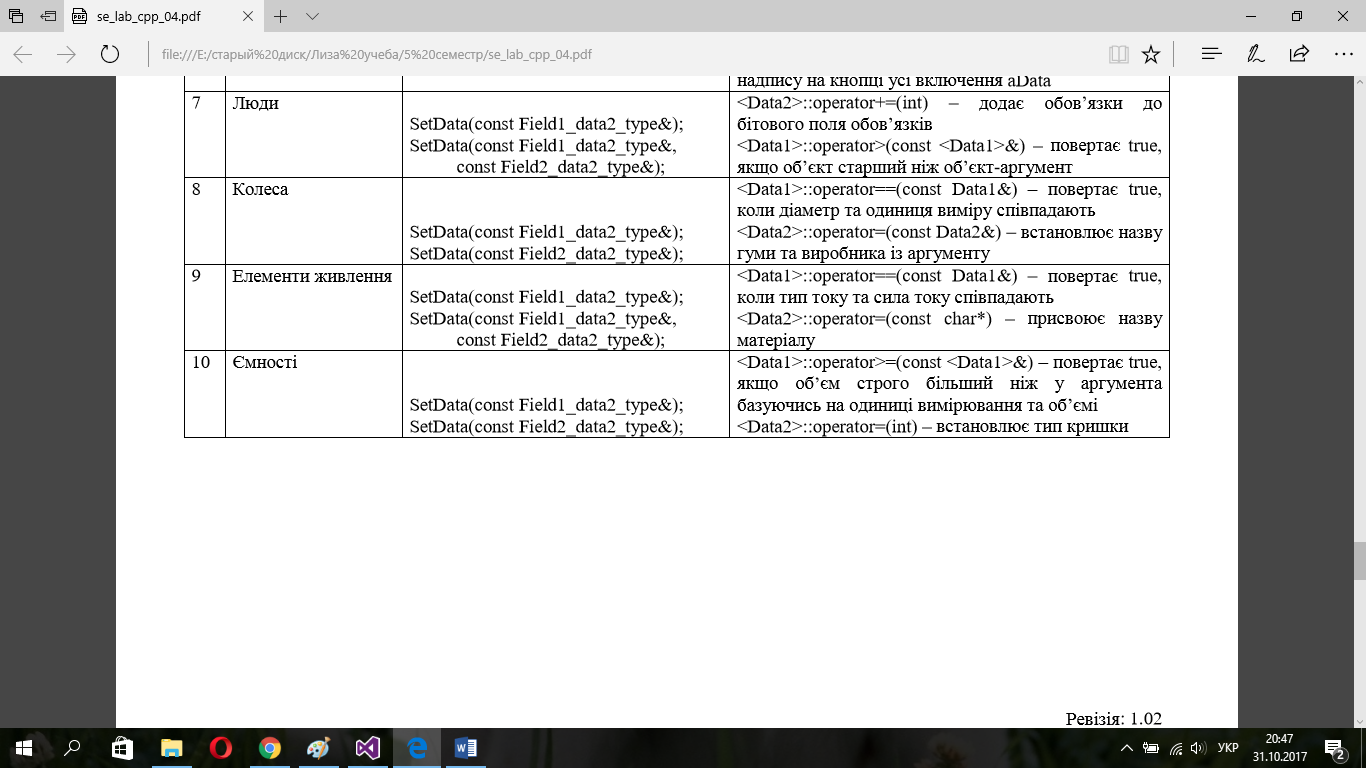
Лабораторна робота №4

Мета:

* Навчитись доречно використовувати статичні методи, а також використовувати перевантаження методів та операторів.

1. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Варіант 10. У класі GraphScreen створити статичний метод onTimerAction(), який відображатиме на екрані заданий нащадок Capacity. Встановити цей метод на виклик у таймері. Таймер повинен спрацювати 4 рази. Метод повинен виводити на екран дані про поточний асоційований об’єкт даних. Реалізувати перевантажені оператори і методи згідно варіанту:



1. РОЗРОБКА ПРОГРАМИ

Для реалізації програми було створено клас Timer та оновлено існуючі класи та методи, згідно індивідуального завдання.

1. Засоби ООП

У розробленій програмі використані наступні засоби ООП:

* розділення програми на ієрархію класів (інкапсуляція);
* поліморфізм;
* спадкування;
* абстракція (віртуальність);

1. Ієрархія та структура класів

На рис.2.2 наведена ієрархія розроблених класів

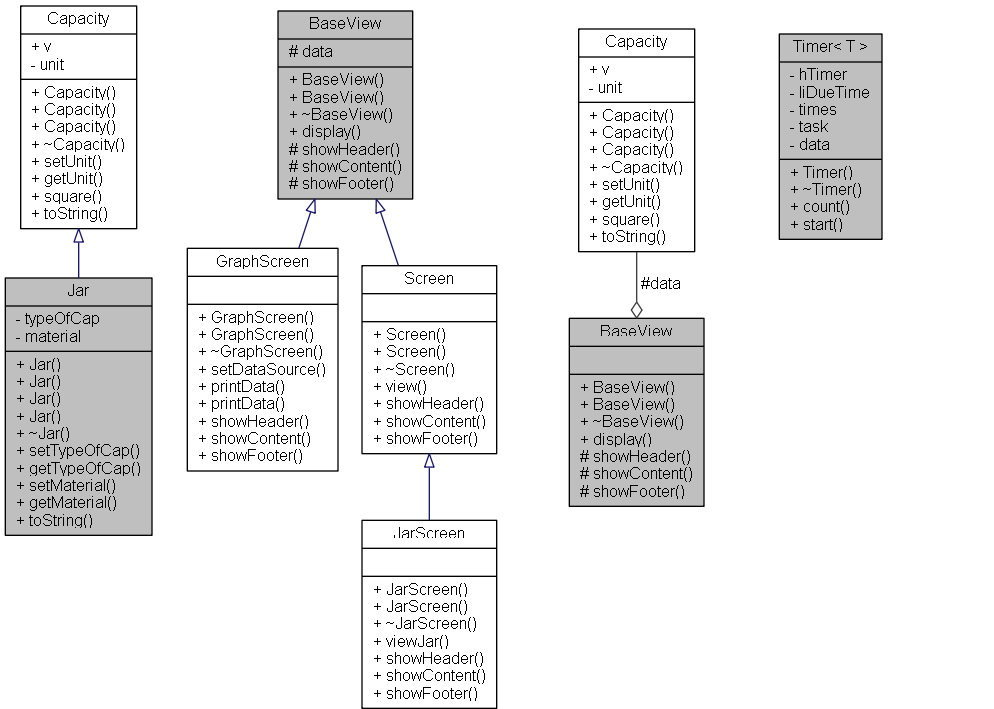


Рисунок 2.2 – Ієрархія класів

1. Опис програми

На рис.2.3 наведена структура розробленого проекту

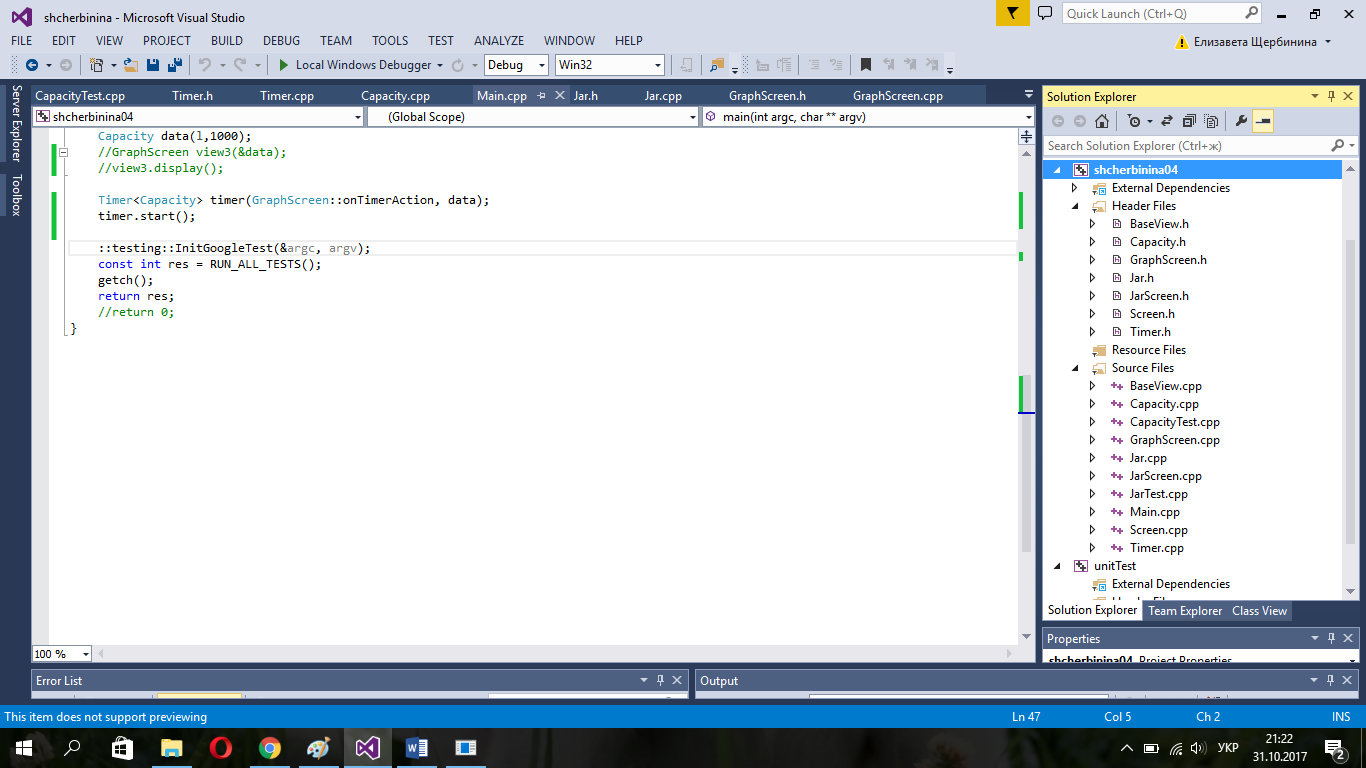


Рисунок 2.3 – Структура проекту

Призначення спроектованих класів наведено на рис.2.4

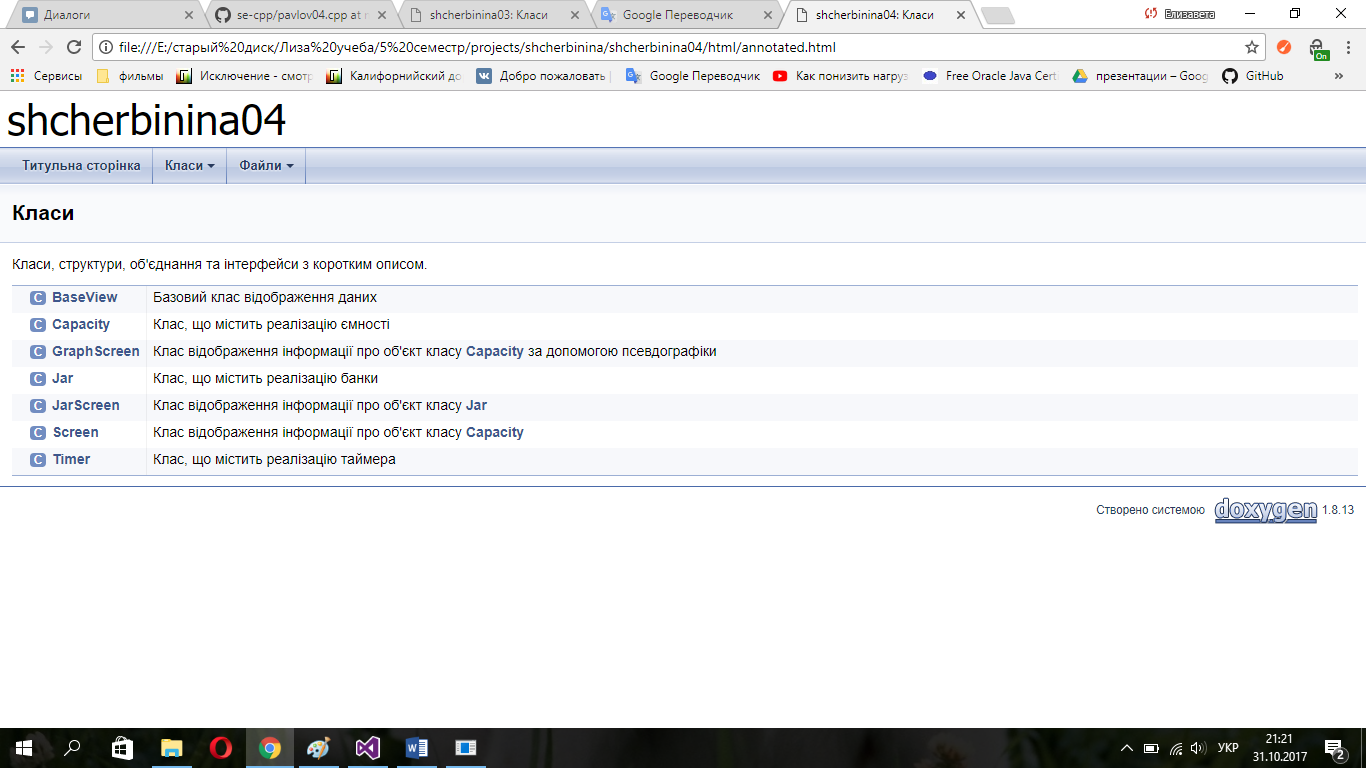


Рисунок 2.4 – Призначення класів

1. Важливі фрагменти програми

Функція відображення об’єкта, що встановлюється у таймері:

///Функція відображення даних

void GraphScreen::onTimerAction(Capacity data){

Screen view((&data));

view.view();

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

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

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^|^^|\n";

cout << " |^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^| |\n";

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | | | |\n";

cout << " | |\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ |\_ |\n";

cout << " |/ | /\n";

cout << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|/\n";

}

Перевантажені методи:

void Jar::setData(string data){

this->setTypeOfCap(data);

}

void Jar::setData(const string& data){

this->setMaterial(data);

}\

Перевантажені оператори:

void Jar::operator = (int data){

switch (data){

case 1:

this->setTypeOfCap("Закручується");

break;

case 2:

this->setTypeOfCap("Закатується");

break;

case 3:

this->setTypeOfCap("Звичайна");

break;

default:

this->setTypeOfCap("Вакуумна");

}

bool Capacity::operator >= (Capacity data){

bool result;

if (this->getUnit() == data.getUnit() || (this->getUnit() == 0 && data.getUnit() == 2) || (this->getUnit() == 2 && data.getUnit() == 1)){

if (this->v >= data.getUnit())

result = true;

else result = false;

}

else if ((this->getUnit() == 2 || this->getUnit() == 0) && data.getUnit() == 1)

result = true;

else result = false;

return result;

}

Клас Timer та його функції:

/\*\*

\* Клас, що містить реалізацію таймера

\*/

template<class T> class Timer

{

private:

HANDLE hTimer;

LARGE\_INTEGER liDueTime;

int times;

void(\*task)(T);

T data;

public:

Timer(void(\*task)(T), T data);

virtual ~Timer();

void count();

void start();

};

template<class T> Timer<T>::Timer(void(\*task)(T), T data) :

task(task), data(data) {

this->times = 4;

this->liDueTime.QuadPart = -10000000LL;

this->hTimer = NULL;

}

template<class T> void Timer<T>::start() {

hTimer = CreateWaitableTimer(NULL, TRUE, L"WaitableTimer");

if (NULL == hTimer) {

printf("CreateWaitableTimer failed (%d)\n", GetLastError());

}

for (int i = 0; i < Timer::times; i++)

count();

}

template<class T> void Timer<T>::count() {

// Set a Timer to wait for 10 seconds.

if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0)) {

printf("SetWaitableTimer failed (%d)\n", GetLastError());

}

// Wait for the Timer.

if (WaitForSingleObject(hTimer, INFINITE) != WAIT\_OBJECT\_0)

printf("WaitForSingleObject failed (%d)\n", GetLastError());

else

task(data);

}

Функція main():

/\*\*

\* Точка входу в програму

\*/

int main(int argc, char \*\*argv) {

setlocale(LC\_ALL, "Russian");

Capacity data(l,1000);

Timer<Capacity> timer(GraphScreen::onTimerAction, data);

timer.start();

::testing::InitGoogleTest(&argc, argv);

const int res = RUN\_ALL\_TESTS();

getch();

return res;

}

1. РЕЗУЛЬТАТИ РОБОТИ

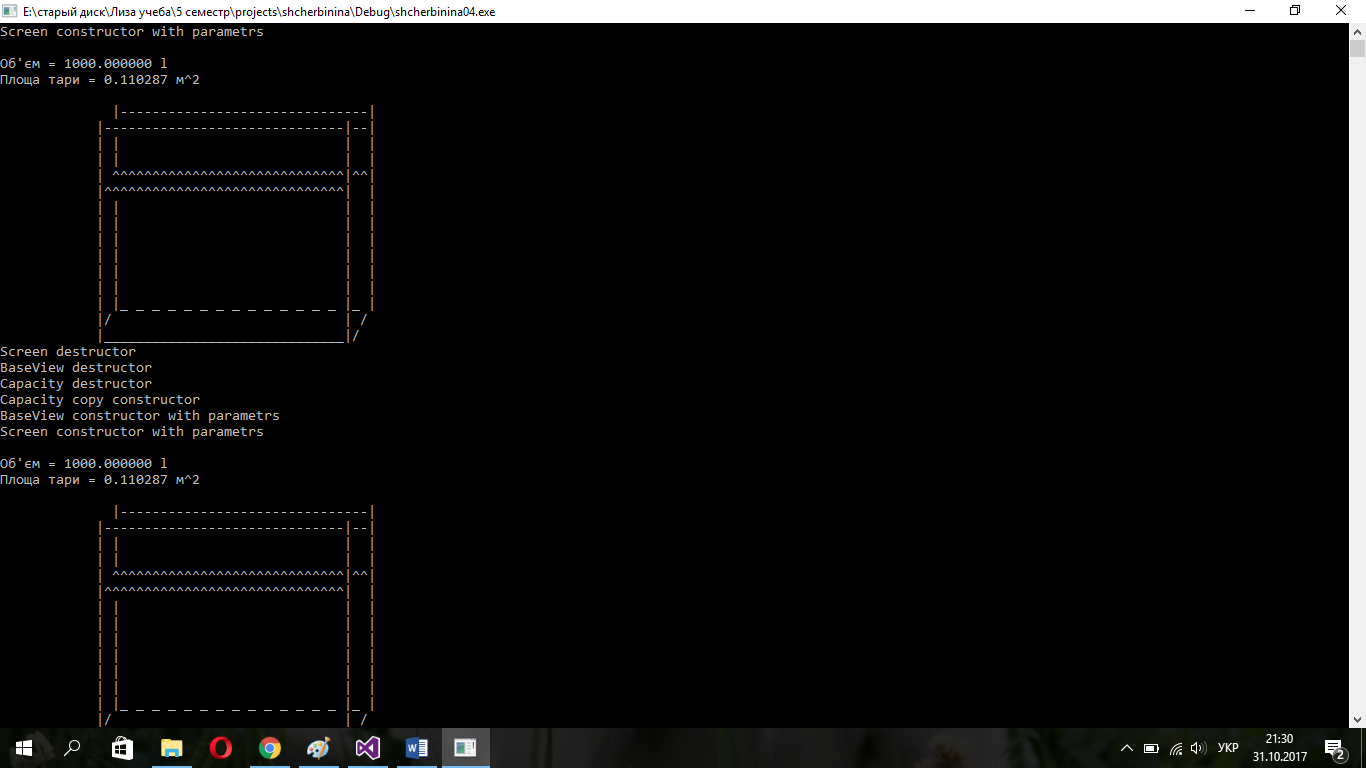


Рисунок 3.1 – Приклад роботи програми

ВИСНОВКИ

В розробленій програмі я отримала навички створення статичних методів та роботи з ними, перевантаження методів там операторів.