АБСТРАКТНІ КЛАСИ, ІНТЕРФЕЙСИ, СЕРІАЛІЗАЦІЯ

Лабораторна робота №5

Мета:

* Навчитись застосовувати інтерфейси для роботи класів на прикладі задачу серіалізації.

1. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Реалізувати для кожного класу даних власної ієрархії можливість збереження та завантаження даних за допомогою класу FileStorage, який видається до лабораторної роботи у вигляді бібліотеки.

Показати у звіті бінарний дамп збереженого файлу та відмітити дані із власних об’єктів.

1. РОЗРОБКА ПРОГРАМИ

Для реалізації програми було оновлено класи даних, що реалізують інтерфейс відповідно до 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

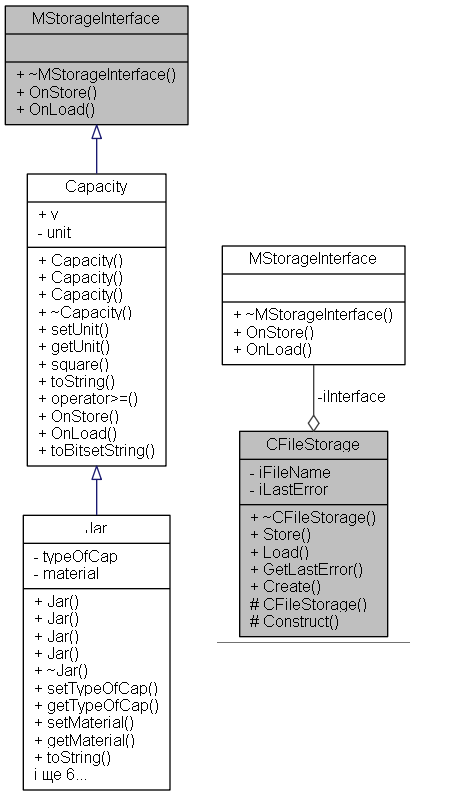


Рисунок 2.2 – Ієрархія класів

1. Опис програми

На рис.2.3 наведена структура розробленого проекту

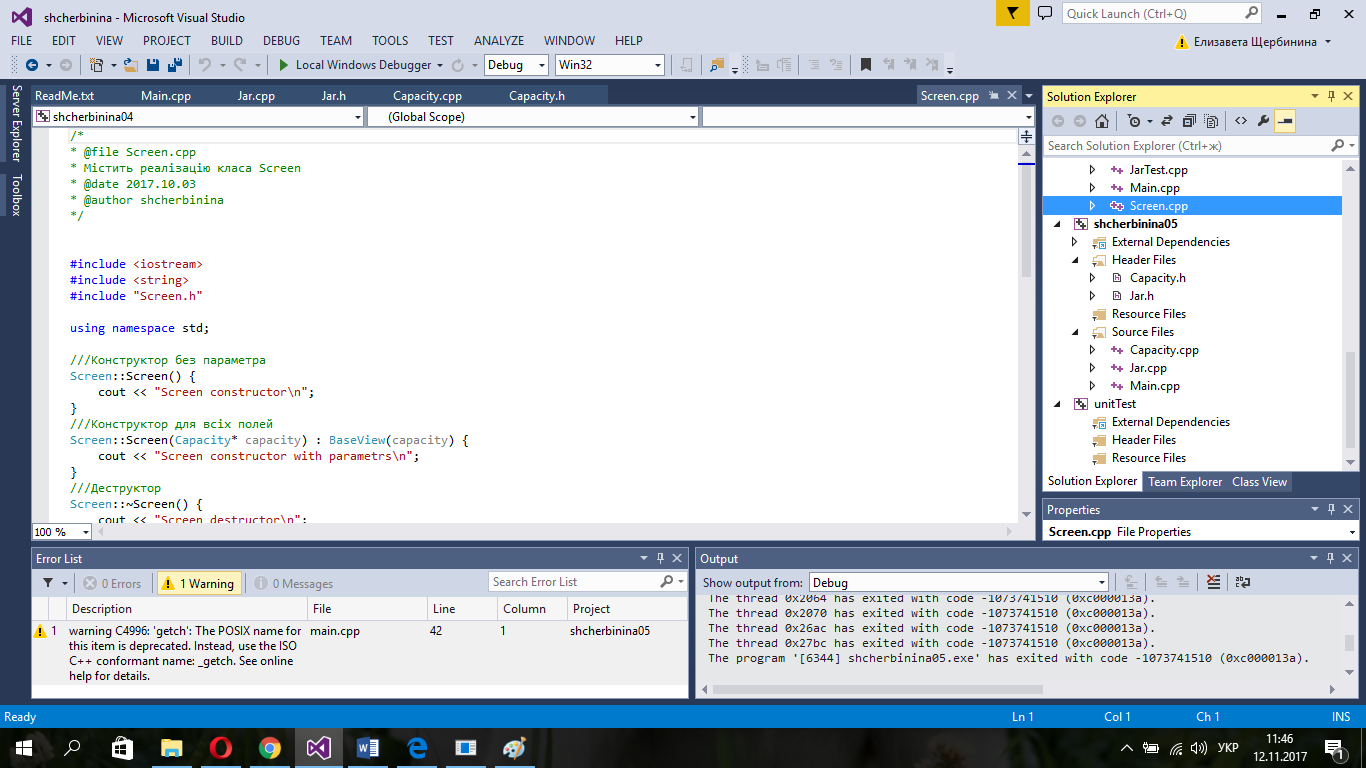


Рисунок 2.3 – Структура проекту

Призначення спроектованих класів наведено на рис.2.4

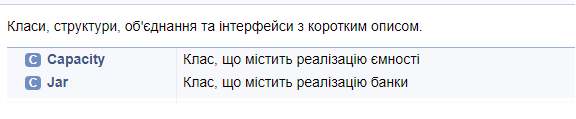


Рисунок 2.4 – Призначення класів

1. Важливі фрагменти програми

Функція переводу у послідовність бітів:

* у класі Capacity

string Capacity::toBitsetString() {

string res;

res += bitset<32>(this->getUnit()).to\_string();

res += bitset<32>(this->v).to\_string();

return res;

}

* у класі Jar

string Jar::toBitsetString(){

string res = Capacity::toBitsetString();

for (unsigned int i = 0; i < this->getMaterial().length(); i++) {

res += bitset<9>(this->getMaterial().at(i)).to\_string();

}

res += " ";

for (unsigned int i = 0; i < this->getTypeOfCap().length(); i++) {

res += bitset<9>(this->getTypeOfCap().at(i)).to\_string();

}

res += " ";

return res;

}

Функція запису у файл:

* у класі Capacity

void Capacity::OnStore(ostream& aStream){

aStream << toBitsetString();

}

* у класі Jar

void Jar::OnStore(ostream& aStream){

aStream << toBitsetString();

}

Функція зчитуванная з файлу:

* у класі Capacity

void Capacity::OnLoad(istream& aStream){

bitset<32> input;

aStream >> input;

this->unit = (units)input.to\_ulong();

aStream >> input;

this->v = (float)input.to\_ulong();

}

* у класі Jar

void Jar::OnLoad(istream& aStream){

Capacity::OnLoad(aStream);

string tmpStr;

bitset<8> input;

while (aStream.get() != ' ') {

aStream >> input;

tmpStr += (char)input.to\_ulong();

}

this->setMaterial(tmpStr);

tmpStr.clear();

while (aStream.get() != ' ') {

aStream >> input;

tmpStr += (char)input.to\_ulong();

}

this->setTypeOfCap(tmpStr);

}

Функція main():

/\*\*

\* Точка входу в програму

\*/

int main() {

setlocale(LC\_ALL, "Russian");

Capacity test(l, 1000);

cout << "Данные перед записью в файл: " << test;

CFileStorage \*testStorage = CFileStorage::Create(test, "Test.bin");

testStorage->Store();

Capacity test2;

testStorage = CFileStorage::Create(test2, "Test.bin");

testStorage->Load();

cout << "Данные, записанные из файла: " << test2;

Jar btest(l, 100, "закручуюється", "пластик");

cout << "Данные перед записью в файл: " << btest.toString();

CFileStorage \*testStorageB = CFileStorage::Create(btest, "Test2.bin");

testStorageB->Store();

Jar btest2;

testStorageB = CFileStorage::Create(btest2, "Test2.bin");

testStorageB->Load();

cout << "Данные, записанные из файла: " << btest2.toString();

getch();

return 0;

}

1. РЕЗУЛЬТАТИ РОБОТИ

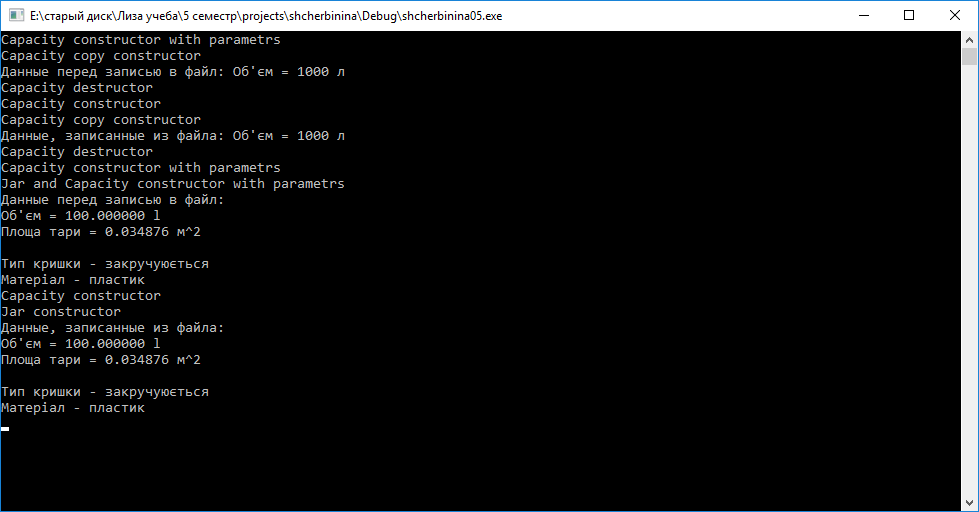


Рисунок 3.1 – Приклад роботи програми

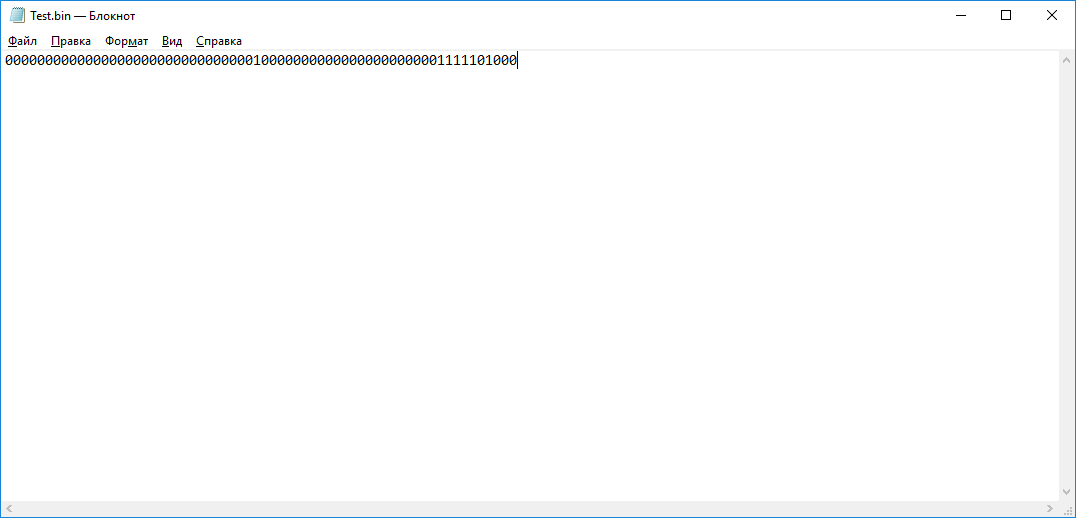


Рисунок 3.2 – Вміст файлу з даними з об’єкту типу Capacity

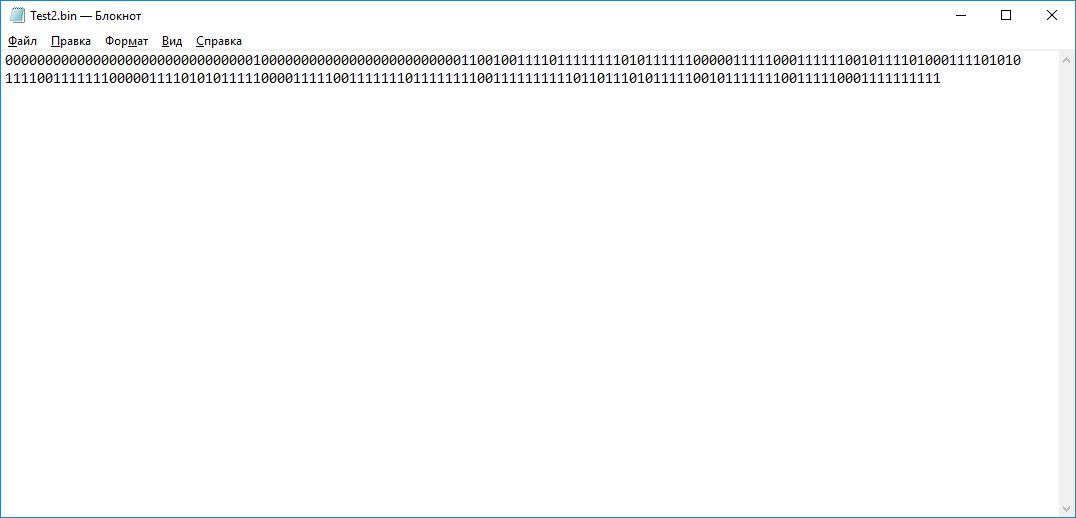


Рисунок 3.3 – Вміст файлу з даними з об’єкту типу Jar

ВИСНОВКИ

В результаті лабораторної роботи було розроблено програму з використанням інтерфейсів. Були виявлені такі недоліки інтерфейсу:

При збереженні об'єкту потрібно перезаписати файл, з попередніми данними, або створити новий файл, нема можливості зберегти у один файл декілька об'єктів. Рішення: створити функцію для збереження масиву, або створити можливість користовачу вибрати, чи хоче він перезаписати дані, або дописати у файл.

Нема перевірки на nullptr у конструкторі, що може привести до помилок. Рішення: перевіряти на nullptr при створенні об'єкту.

При запису у файл функція OnStore приймає поток запису, при цьому класи реалізуючі інтерфейс MStorageInterface мають доступ до нього, та можуть пошкодити файл, або змінити його на nullptr. Рішення: замінити OnStore на функцію, яка буде повертати дані об'єкту у потрібному реалізатору вигляді, та записувати у методі Store у файл результат нової функції.