Міністерство освіти і науки, молоді та спорту України
Національний університет «Львівська політехніка»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

Про виконання лабораторної роботи № 8

«Наслідування. Створення та використання ієрархії класів.»
з дисципліни «Об'єктно-орієнтоване програмування»

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студ. групи ПЗ-15
Марущак А.С.

Прийняв:
доцент кафедри ПЗ
Яцишин С.І.
«___» _____ 2022 р.
∑ = ____ _____

Львів – 2022

Тема роботи: Наслідування. Створення та використання ієрархії класів.

Мета роботи: Навчитися створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувати принципи використання множинного наслідування. Навчитися перевизначати методи в похідному класі, освоїти принципи такого перевизначення.

Теоретичні відомості

Наслідування

Наслідуванням називається процес визначення класу на основі іншого класу. На новий (дочірній) клас за замовчуванням поширюються всі визначення змінних екземпляра і методів зі старого (батьківського) класу, але можуть бути також визначені нові компоненти або «перевизначені» визначення батьківських функцій і дано нові визначення. Прийнято вважати, що клас А успадковує свої визначення від класу В, якщо клас А визначений на основі класу В зазначеним способом.

Класи можуть бути пов'язані один з одним різними відношеннями. Одним з основних є відношення *клас-підклас*, відоме в об'єктно-орієнтованому програмуванні як *наслідування*. Наприклад, клас автомобілів Audi 6 є підкласом легкових автомобілів, який в свою чергу входить у більший клас автомобілів, а останній є підкласом класу транспортних засобів, який крім автомобілів включає в себе літаки, кораблі, потяги і т.д. Прикладом подібних відношень є системи класифікації в ботаніці та зоології.

При наслідуванні всі атрибути і методи батьківського класу успадковуються **класом-нащадком**. Наслідування може бути багаторівневим, і тоді класи, що знаходяться на нижніх рівнях ієрархії, успадкують всі властивості (атрибути і методи) всіх класів, прямими або непрямими нащадками яких вони є.

Крім одиничного, існує і **множинне** наслідування, коли клас наслідує відразу кілька класів. При цьому він успадкує властивості всіх класів, нащадком яких він є.

При наслідуванні одні методи класу можуть заміщатися іншими. Так, клас транспортних засобів буде мати узагальнений метод руху. У класах-нащадках цей метод буде конкретизований: автомобіль буде їздити, літак – літати, корабель – плавати. Така зміна семантики методу називається *поліморфізмом*. **Поліморфізм** – це виконання методом з одним і тим же ім'ям різних дій залежно від контексту, зокрема, від приналежності до того чи іншого класу. У різних мовах програмування поліморфізм реалізується різними способами.

Синтаксис наслідування класів:

```
class Animal

{
```

```
private:
    int age;
public:
    int GetAge( )
    {
        return age;
    }
};


class Horse : public Animal
{
public:
    virtual void Gallop(){ cout << "Gallop !!!"; }
};
```

Заміщення функцій.

Крім цього базові функції можуть бути заміщені в похідному класі. Під заміщенням базової функції розуміють зміну її виконання в похідному класі.

Для заміщення необхідно описати функцію в похідному класі з таким же ж іменем як у базовому.

Недоліки заміщення.

Якщо в базовому класі у нас є перевантажена функція, яку ми хочемо замінити в похідному класі – ми не зможемо викликати в похідному класі будь-яку з цих перевантажених функцій:

Індивідуальне завдання

1. Розробити ієрархію класів відповідно до варіанту
2. Створити базовий, похідні класи.
3. Використати public, protected наслідування.

4. Використати множинне наслідування (за необхідності).
5. Виконати перевантаження функції print() в базовому класі, яка друкує назву відповідного класу, перевизначити її в похідних. В проекті при натисканні кнопки виведіть на форму назви всіх розроблених класів.
6. Реалізувати методи варіанта та результати вивести на **форму** і у **файл**. При записі у файл використати різні варіанти **аргументів конструктора**.
7. Оформити звіт до лабораторної роботи. Включити у звіт **Uml-діаграму** розробленої ієрархії класів.

Варіант завдання

6. Розробити ієрархію класів для сутності: **поштове відправлення**.

Розробити наступні типи відправлень

- Бандероль звичайна;
- Бандероль із оголошеною цінністю;
- Електронний переказ.

Класи повинні мати повний набір методів для роботи з ними.

Кожен клас обов’язково повинен вміти обчислити вартість відправлення

Хід роботи

Спроектуємо ієрархію класів за допомогою uml-діаграми:

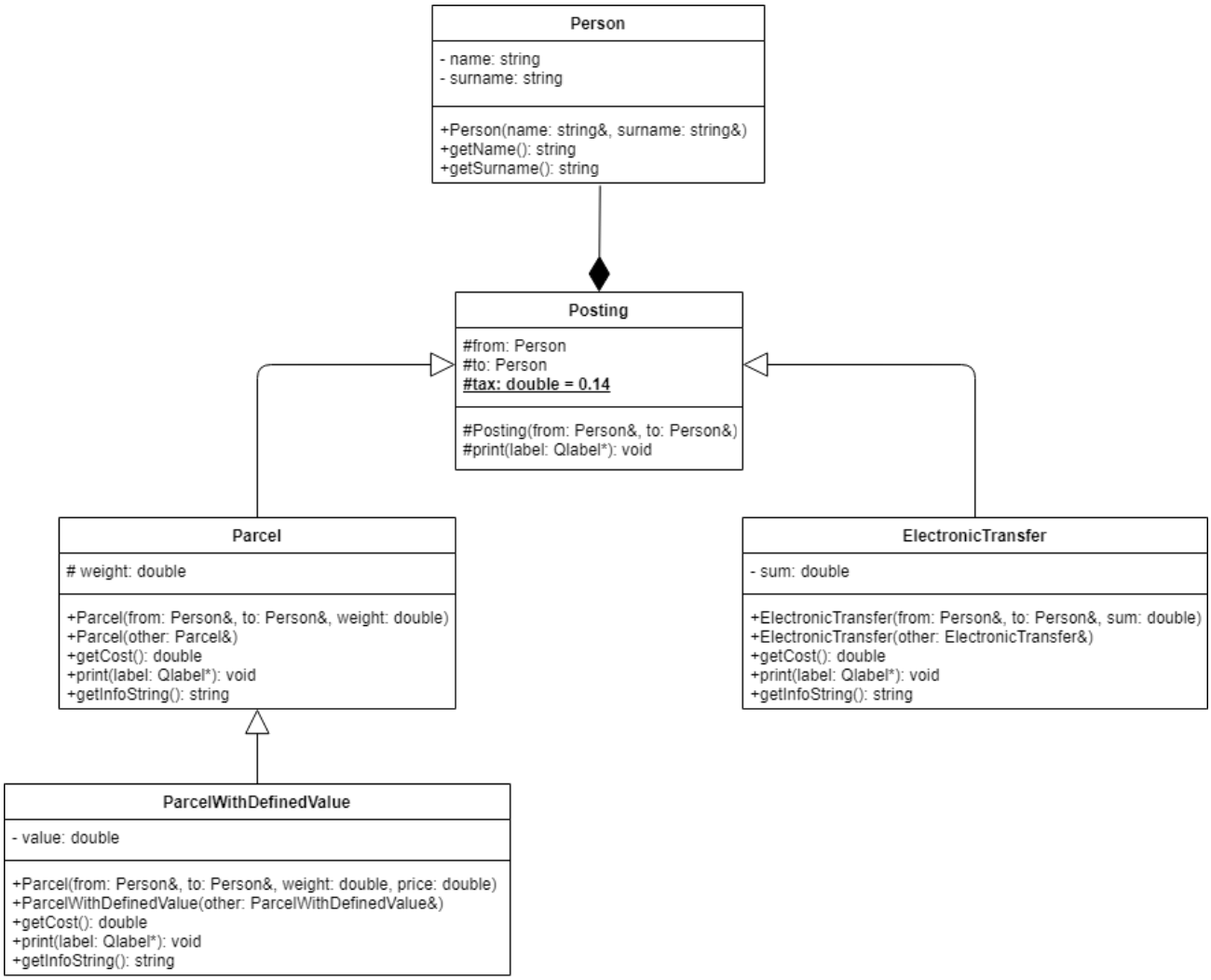


Рис 8.1 UML-діаграма класів

Далі опишемо всі ці класи в хедер-файлах(для зручності в звіті зробимо описи класів разом):

```
#include <string>
class Person
{
    const std::string name;
    const std::string surname;
public:
    Person(const std::string& name, const std::string& surname);
    std::string getName() const;
    std::string getSurname() const;
};


#include <QLabel>

class Posting
{
protected:
    static constexpr double tax = 0.14;
    const Person from;
    const Person to;
    Posting(const Person& from, const Person& to);
```

```
    void print(QLabel* label) const;
};


class Parcel : protected Posting
{
protected:
    const double weight;
public:
    Parcel(const Person& from, const Person& to, const double weight);
    Parcel(const Parcel& other);
    double getCost() const;
    void print(QLabel* label) const;
    std::string getInfoString() const;

};

class ParcelWithDefinedValue : public Parcel
{
    const double value;
public:
    ParcelWithDefinedValue(const Person &from, const Person &to, const double weight,
const double value);
    ParcelWithDefinedValue(const ParcelWithDefinedValue& other);
    double getCost() const;
    void print(QLabel* label) const;
    std::string getInfoString() const;
};

class ElectronicTransfer : protected Posting
{
    const double sum;
public:
    ElectronicTransfer(const Person& from, const Person& to, const double sum);
    ElectronicTransfer(const ElectronicTransfer& other);
    double getCost() const;
    void print(QLabel* label) const;
    std::string getInfoString() const;
};
```

І після цього реалізуємо подані класи:

```
Person::Person(const std::string &name, const std::string &surname) : name(name),
surname(surname)
{}

std::string Person::getName() const {return name;}
std::string Person::getSurname() const {return surname;}

Posting::Posting(const Person &from, const Person &to) : from(from), to(to)
{}

void Posting::print(QLabel *label) const
{
    label->setText("Used class: Posting");
}
```

```
Parcel::Parcel(const Person &from, const Person &to, const double weight) :
Posting(from, to), weight(std::max(100.0, weight))
{}

Parcel::Parcel(const Parcel &other) : Posting(other.from, other.to),
weight(other.weight)
{}

double Parcel::getCost() const
{
    if(weight <= 250) return 22.5 * (1+tax);
    else if(weight < 1000) return 45 * (1+tax);
    else if(weight < 2000) return 60 * (1+tax);
    else return 75 * (1+tax);
}

void Parcel::print(QLabel *label) const
{
    label->setText("Used class: Parcel");
}

std::string Parcel::getInfoString() const
{
    return "Parcel from " + from.getName() + " " + from.getSurname() + " to " +
to.getName() + " " + to.getSurname() + " with the price " + std::to_string(getCost()) +
" UAH";
}

ParcelWithDefinedValue::ParcelWithDefinedValue(const Person &from, const Person &to,
const double weight, const double value) : Parcel(from, to, weight), value(value)
{}

ParcelWithDefinedValue::ParcelWithDefinedValue(const ParcelWithDefinedValue &other) :
Parcel(other.from, other.to, other.weight), value(other.value)
{}

double ParcelWithDefinedValue::getCost() const
{
    return Parcel::getCost() + std::min(1.0, 0.01 * value) * (1 + tax);
}

void ParcelWithDefinedValue::print(QLabel *label) const
{
    label->setText("Used class: ParcelWithDefinedValue");
}

std::string ParcelWithDefinedValue::getInfoString() const
{
    return "Parcel with defined value from " + from.getName() + " " + from.getSurname()
+ " to " + to.getName() + " " + to.getSurname() + " with the price " +
std::to_string(getCost()) + " UAH";
}

ElectronicTransfer::ElectronicTransfer(const Person &from, const Person &to, const
double sum) : Posting(from, to), sum(sum)
{}

ElectronicTransfer::ElectronicTransfer(const ElectronicTransfer &other) :
Posting(other.from, other.to), sum(other.sum)
{}

double ElectronicTransfer::getCost() const
{
    if(sum < 2000) return (0.02 * sum) * (1 + tax);
```

```
    else return (0.015 * sum) * (1 + tax);
}

void ElectronicTransfer::print(QLabel *label) const
{
    label->setText("Used class: ParcelWithDefinedValue");
}

std::string ElectronicTransfer::getInfoString() const
{
    return "Electronic transfer from " + from.getName() + " " + from.getSurname() + "
to " + to.getName() + " " + to.getSurname() + " with the price " +
std::to_string(getCost()) + " UAH";
}
```

Для демонстрації можливостей цих класів створимо віконний додаток з наступним інтерфейсом:

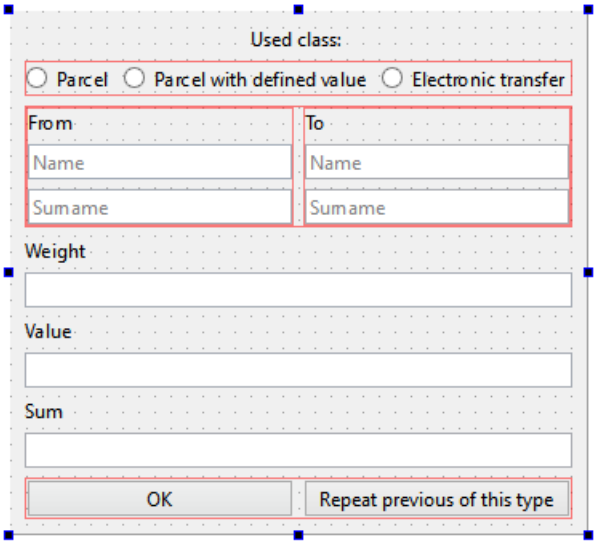


Рис 8.2 Дизайн вікна

І реалізуємо його методи наступним чином:

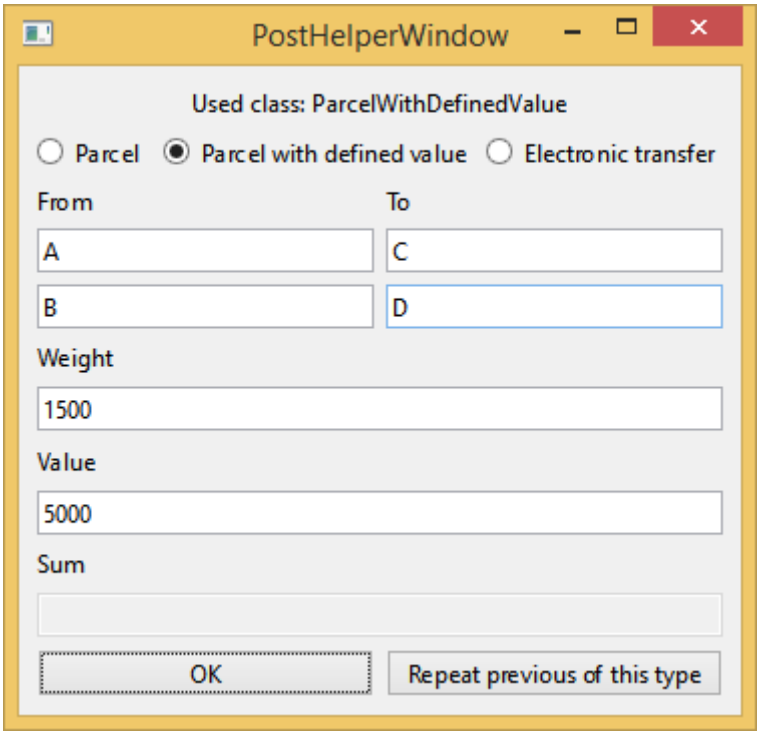
```
#include "posthelperwindow.h"
#include "ui_posthelperwindow.h"
#include <QString>
#include <QMessageBox>

PostHelperWindow::PostHelperWindow(QWidget *parent)
    : QMainWindow(parent)
    , stream("postings.txt", std::ios::app)
    , ui(new Ui::PostHelperWindow)
{
    ui->setupUi(this);

    ui->parcelRadioButton->setChecked(true);
    on_parcelRadioButton_clicked(true);
}

PostHelperWindow::~PostHelperWindow()
{
    delete ui;
    delete lastParcel;
    delete lastElectronicTransfer;
    delete lastParcelWithDefinedValue;
    stream.close();
```

```
}


void PostHelperWindow::on_parcelRadioButton_clicked(bool checked)
{
    if(checked)
    {
        ui->weightEdit->setDisabled(false);
        ui->valueEdit->setDisabled(true);
        ui->sumEdit->setDisabled(true);
    }
}


void PostHelperWindow::on_valueParcelRadioButton_clicked(bool checked)
{
    if(checked)
    {
        ui->weightEdit->setDisabled(false);
        ui->valueEdit->setDisabled(false);
        ui->sumEdit->setDisabled(true);
    }
}


void PostHelperWindow::on_electronicTranferRadioButton_clicked(bool checked)
{
    if(checked)
    {
        ui->weightEdit->setDisabled(true);
        ui->valueEdit->setDisabled(true);
        ui->sumEdit->setDisabled(false);
    }
}


void PostHelperWindow::on_confirmationButton_clicked()
{
    if(ui->parcelRadioButton->isChecked())
    {
        Parcel* toOutput = new Parcel
                (
                    Person(ui->fromNameEdit->text().toStdString(), ui->fromSurnameEdit-
>text().toStdString()),
                    Person(ui->toNameEdit->text().toStdString(), ui->toSurnameEdit-
>text().toStdString()),
                    ui->weightEdit->text().toDouble()
                );

        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput->getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            delete lastParcel;
            lastParcel = toOutput;
            stream << toOutput->getInfoString() << "\n";
            stream.flush();
            toOutput->print(ui->usedClassLabel);
        }
    }
    else if(ui->valueParcelRadioButton->isChecked())
    {
        ParcelWithDefinedValue* toOutput = new ParcelWithDefinedValue
```

```
                (
                    Person(ui->fromNameEdit->text().toStdString(), ui->fromSurnameEdit-
>text().toStdString()),
                    Person(ui->toNameEdit->text().toStdString(), ui->toSurnameEdit-
>text().toStdString()),
                    ui->weightEdit->text().toDouble(),
                    ui->valueEdit->text().toDouble()
                );

        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput->getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            delete lastParcelWithDefinedValue;
            lastParcelWithDefinedValue = toOutput;
            stream << toOutput->getInfoString() << "\n";
            stream.flush();
            toOutput->print(ui->usedClassLabel);
        }
    }
    else
    {
        ElectronicTransfer* toOutput = new ElectronicTransfer
                (
                    Person(ui->fromNameEdit->text().toStdString(), ui->fromSurnameEdit-
>text().toStdString()),
                    Person(ui->toNameEdit->text().toStdString(), ui->toSurnameEdit-
>text().toStdString()),
                    ui->sumEdit->text().toDouble()
                );

        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput->getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            delete lastElectronicTransfer;
            lastElectronicTransfer = toOutput;
            stream << toOutput->getInfoString() << "\n";
            stream.flush();
            toOutput->print(ui->usedClassLabel);
        }
    }
}


void PostHelperWindow::on_pushButton_clicked()
{
    if(ui->parcelRadioButton->isChecked() && lastParcel)
    {
        Parcel toOutput(*lastParcel);
        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput.getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            stream << toOutput.getInfoString() << "\n";
            stream.flush();
            toOutput.print(ui->usedClassLabel);
        }
    }
    else if(ui->valueParcelRadioButton->isChecked() && lastParcelWithDefinedValue)
    {
```

```
        ParcelWithDefinedValue toOutput(*lastParcelWithDefinedValue);
        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput.getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            stream << toOutput.getInfoString() << "\n";
            stream.flush();
            toOutput.print(ui->usedClassLabel);
        }
    }
    else if(lastElectronicTransfer)
    {
        ElectronicTransfer toOutput(*lastElectronicTransfer);
        QMessageBox::StandardButton btn = QMessageBox::information(this, "New posting",
QString::fromStdString(toOutput.getInfoString()), QMessageBox::Ok,
QMessageBox::Cancel);
        if(btn == QMessageBox::Ok)
        {
            stream << toOutput.getInfoString() << "\n";
            stream.flush();
            toOutput.print(ui->usedClassLabel);
        }
    }
}
```

Результат виконання:



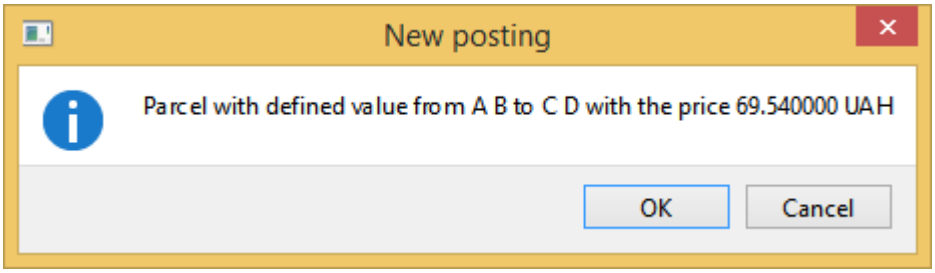


Рис 8.3-4 Результат виконання програми.

Висновок: виконавши лабораторну роботу №8, ми навчилися створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанували принципи використання множинного наслідування. Навчилися перевизначати методи в похідному класі, освоїли принципи такого перевизначення. Робота над лабораторною роботою була плідною та ефективною.