

Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій

Кафедра ПЗ

Звіт

до лабораторної роботи №5
на тему «Створення та використання класів»
з дисципліни “Об’єктно-орієнтоване програмування”

Виконав:

студент групи ПЗ-11

Солтисюк Дмитро

Перевірив:

доц. Коротєєва Т.О.

Львів

2022

Тема. Створення та використання класів.

Мета. Навчитися створювати класи, використовувати конструктори для ініціалізації об'єктів, опанувати принципи створення функцій-членів. Навчитися використовувати різні типи доступу до полів та методів класів.

Варіант 24 (трикутник)

Завдання для лабораторної роботи:

1. Створити клас відповідно до варіанту.
2. При створенні класу повинен бути дотриманий принцип інкапсуляції.
3. Створити конструктор за замовчуванням та хоча б два інших конструктори для початкової ініціалізації об'єкта.
4. Створити функції члени згідно з варіантом.
5. Продемонструвати можливості класу завдяки створеному віконному застосуванню.
6. У звіті до лабораторної намалювати UML-діаграму класу, яка відповідає варіанту.

Клас Triangle – трикутник на площині (задаються довжини трьох сторін). Клас повинен містити функції-члени, які реалізують: а) Знаходження площі трикутника б) Знаходження трьох кутів в) Знаходження периметра г) Знаходження трьох висот д) Збільшення одразу всіх трьох сторін трикутника на константу е) Задавання значень полів є) Зчитування (отримання значень полів) ж) Перевірка чи трикутник є прямокутний з) Введення трикутника з форми и) Виведення трикутника на форму.

Теоретичні відомості:

Ідея класів має на меті дати інструментарій для відображення будови об'єктів реального світу - оскільки кожен предмет або процес має набір характеристик (відмінних рис) іншими словами, володіє певними властивостями і поведінкою. Програми часто призначені для моделювання предметів, процесів і явищ реального світу, тому в мові програмування зручно мати адекватний інструмент для представлення цих моделей.

Клас є типом даних, який визначається користувачем. У класі задаються властивості і поведінка будь-якого предмету або процесу у вигляді полів даних (аналогічно до того як це є в структурах) і функцій для роботи з ними. Створюваний тип даних володіє практично тими ж властивостями, що і стандартні типи.

Конкретні величини типу даних «клас» називаються екземплярами класу, або об'єктами.

Об'єднання даних і функцій їх обробки з одночасним приховуванням непотрібної для використання цих даних інформації називається інкапсуляцією (encapsulation). Інкапсуляція підвищує ступінь абстракції програми: дані класу і реалізація його функцій знаходяться нижче рівня абстракції, і для написання програми з використанням вже готових класів інформації про них (дані і реалізацію функцій) не потрібно. Крім того, інкапсуляція дозволяє змінити

реалізацію класу без модифікації основної частини програми, якщо інтерфейс залишився тим самим (наприклад, при необхідності змінити спосіб зберігання даних з масиву на стек). Простота модифікації, як уже неодноразово зазначалося, є дуже важливим критерієм якості програми.

Опис класу в першому наближенні виглядає так:

```
class <ім'я> {  
[private:]  
<Опис прихованих елементів>  
public:  
<Опис доступних елементів>  
}; //Опис закінчується крапкою з комою.
```

Специфікатор доступу `private` і `public` керують видимістю елементів класу. Елементи, описані після службового слова `private`, видимі тільки всередині класу. Цей вид доступу прийнятий у класі за замовчуванням. Інтерфейс класу описується після специфікатора `public`. Дія будь-якого специфікатора поширюється до наступного специфікатора або до кінця класу. Можна задавати кілька секцій `private` і `public`, їх порядок значення не має.

Поля класу:

- можуть мати будь-який тип, крім типу цього ж класу (але можуть бути вказівниками або посиланнями на цей клас);
- можуть бути описані з модифікатором `const`, при цьому вони ініціалізуються тільки один раз (за допомогою конструктора) і не можуть змінюватися;
- можуть бути описані з модифікатором `static` (розглядається в наступних лабораторних).

Ініціалізація полів при описі не допускається.

Конструктори.

Конструктор призначений для ініціалізації об'єкту і викликається автоматично при його створенні. Автоматичний виклик конструктора дозволяє уникнути помилок, пов'язаних з використанням неініціалізованих змінних. Нижче наведені основні властивості конструкторів:

- Конструктор не повертає жодного значення, навіть типу `void`. Неможливо отримати вказівник на конструктор.
- Клас може мати декілька конструкторів з різними параметрами для різних видів ініціалізації (при цьому використовується механізм перевантаження).
- Конструктор без параметрів називається конструктором за замовчуванням.
- Параметри конструктора можуть мати будь-який тип, крім цього ж класу.

Можна задавати значення параметрів за замовчуванням. Їх може містити тільки один з конструкторів.

• Якщо програміст не вказав жодного конструктора, компілятор створює його автоматично. Такий конструктор викликає конструктори за замовчуванням для полів класу і конструктори за замовчуванням базових класів. У разі, коли клас містить константи або посилання, при спробі створення об'єкту класу буде видана помилка, оскільки їх необхідно ініціалізувати конкретними значеннями, а конструктор за замовчуванням цього робити не вміє.

- Конструктори не наслідуються.
- Конструктори не можна описувати з модифікаторами `const`, `virtual` і `static`.
- Конструктори глобальних об'єктів викликаються до виклику функції `main`.

Локальні об'єкти створюються, як тільки стає активною область їх дії.

Конструктор запускається і при створенні тимчасового об'єкта (наприклад, при передачі об'єкта з функції).

- Конструктор викликається, якщо в програмі зустрілася будь-яка із синтаксичних конструкцій:

```
ім'я_класу ім'я_об'єкту [(список параметрів)];
```

```
//Список параметрів не повинен бути порожнім
```

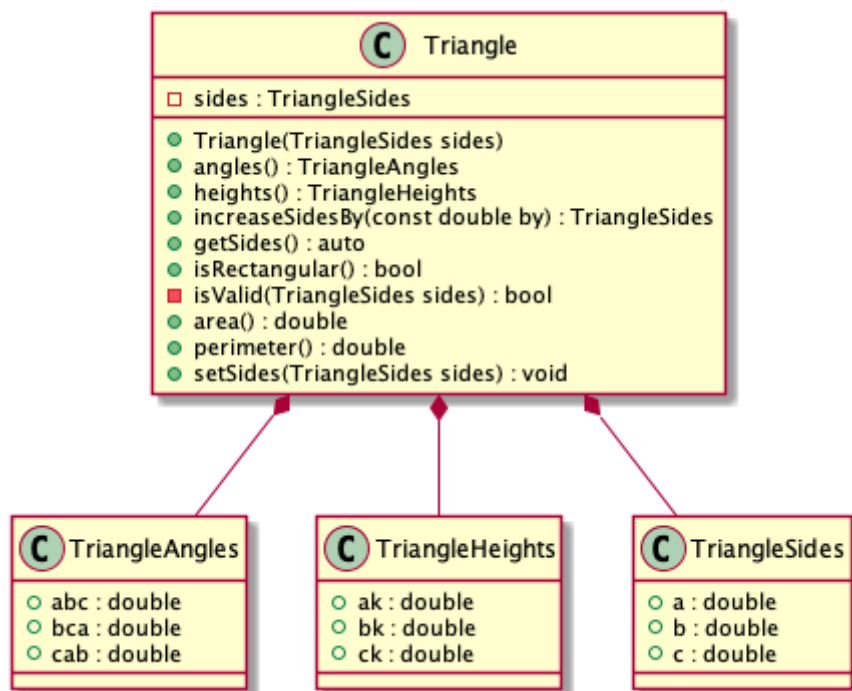
```
ім'я_класу (список параметрів);
```

```
//Створюється об'єкт без імені (список може бути //порожнім)
```

```
ім'я_класу ім'я_об'єкту = вираз;
```

```
//Створюється об'єкт без імені і копіюється
```

Результат:



trianglesides.h

```
#pragma once
```

```
#include "triangleangles.h"
```

```
#include "triangleheights.h"
```

```
#include "trianglesides.h"
```

triangleangles.h

```
#pragma once

class TriangleAngles {
public:
    double abc;
    double bca;
    double cab;
};
```

triangleheights.h

```
#pragma once

class TriangleHeights {
public:
    double ak;
    double bk;
    double ck;
};
```

triangle.h

```
#pragma once

#include "triangleangles.h"
#include "triangleheights.h"
#include "trianglesides.h"

class Triangle {
private:
    TriangleSides sides;
    bool isValid(TriangleSides sides);

public:
    auto getSides() { return this->sides; }
    void setSides(TriangleSides sides) { this->sides = sides; };

    Triangle(TriangleSides sides);

    bool isRectangular();
    double area();
    double perimeter();
    TriangleAngles angles();
    TriangleHeights heights();
    TriangleSides increaseSidesBy(const double by);
};
```

triangle.cpp

```
#include "triangle.h"
#include "triangleangles.h"
```

```

#include "trianglesides.h"
#include <algorithm>
#include <cmath>
#include <iostream>
#include <stdexcept>

bool Triangle::isValid(TriangleSides sides) {
    const auto biggestSide = std::max({sides.a, sides.b, sides.c});
    const auto shortenedSum = sides.a + sides.b + sides.c - biggestSide;

    if (shortenedSum > biggestSide) {
        return true;
    }

    return false;
}

Triangle::Triangle(TriangleSides sides) {
    if (!this->isValid(sides)) {
        throw std::invalid_argument("Received triangle cannot exist");
    }

    this->setSides(sides);
}

bool Triangle::isRectangular() {
    return pow(this->sides.c, 2) == pow(this->sides.a, 2) + pow(this->sides.b, 2);
};

double Triangle::perimeter() {
    return this->sides.a + this->sides.b + this->sides.c;
}

double Triangle::area() {
    // half-perimeter
    double p = this->perimeter() / 2;

    // hero's formula
    return sqrt(p * (p - this->sides.a) * (p - this->sides.b) *
                (p - this->sides.c));
}

TriangleAngles Triangle::angles() {
    TriangleAngles angles;

    const auto angleFinder = [](const double a, const double b, const double c) {
        return (180. / M_PI) *
            acos((pow(a, 2) + pow(b, 2) - pow(c, 2)) / (2 * a * b));
    };

    angles.cab = angleFinder(this->sides.a, this->sides.b, this->sides.c);
    angles.abc = angleFinder(this->sides.a, this->sides.c, this->sides.b);
    angles.bca = angleFinder(this->sides.b, this->sides.c, this->sides.a);

    return angles;
}

TriangleHeights Triangle::heights() {
    TriangleHeights heights;

```

```

    auto area = this->area();

    heights.ak = (2 * area) / this->sides.a;
    heights.bk = (2 * area) / this->sides.b;
    heights.ck = (2 * area) / this->sides.c;

    return heights;
}

TriangleSides Triangle::increaseSidesBy(const double by) {
    this->sides.a += by;
    this->sides.c += by;
    this->sides.b += by;

    return this->sides;
}

```

widget.h

```

#ifndef WIDGET_H
#define WIDGET_H

#include "triangle.h"

#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QTextEdit>
#include <QWidget>

class Widget : public QWidget {
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);

private slots:
    void onInputConfirm();
    void onInputIncrease();
    void onValueChange(Triangle *triangle);

signals:
    void valueChanged(Triangle *triangle);

private:
    QLineEdit *side_a;
    QLineEdit *side_b;
    QLineEdit *side_c;
    QPushButton *confirmInput;

    QLineEdit *increaseSidesBy;
    QPushButton *confirmIncrease;

    QLineEdit *area;
    QLineEdit *perimeter;
    QLineEdit *isRectangular;
    QTextEdit *angles;
    QTextEdit *heights;

```

```

    Triangle *triangle;
};
#endif // WIDGET_H

```

widget.cpp

```

#include "widget.h"
#include "triangle.h"
#include "triangleangles.h"
#include "trianglesides.h"

#include <QGridLayout>
#include <QMessageBox>

void Widget::onInputConfirm() {
    TriangleSides sides;

    sides.a = this->side_a->text().toDouble();
    sides.b = this->side_b->text().toDouble();
    sides.c = this->side_c->text().toDouble();

    this->triangle = new Triangle(sides);

    emit valueChanged(this->triangle);
}

void Widget::onInputIncrease() {
    this->triangle->increaseSidesBy(this->increaseSidesBy->text().toDouble());

    emit valueChanged(this->triangle);
}

void Widget::onValueChange(Triangle *triangle) {
    TriangleSides sides = triangle->getSides();
    TriangleHeights heights = triangle->heights();
    TriangleAngles angles = triangle->angles();

    this->side_a->setText(QString::number(sides.a));
    this->side_b->setText(QString::number(sides.b));
    this->side_c->setText(QString::number(sides.c));

    this->area->setText(QString::number(triangle->area()));
    this->perimeter->setText(QString::number(triangle->perimeter()));
    this->isRectangular->setText(QVariant(triangle->isRectangular()).toString());

    this->heights->setText(QString("AK: %1\nBK: %2\nCK: %3")
        .arg(heights.ak)
        .arg(heights.bk)
        .arg(heights.ck));

    this->angles->setText(QString("ABC: %1\nBCA: %2\nCAB: %3")
        .arg(angles.abc)
        .arg(angles.bca)
        .arg(angles.cab));
}

Widget::Widget(QWidget *parent) : QWidget(parent) {
    QGridLayout *mainLayout = new QGridLayout;

```



```

this->triangle = nullptr;

this->side_a = new QLineEdit;
this->side_a->setPlaceholderText("Side A length");

this->side_b = new QLineEdit;
this->side_b->setPlaceholderText("Side B length");

this->side_c = new QLineEdit;
this->side_c->setPlaceholderText("Side C length");

this->increaseSidesBy = new QLineEdit;

this->isRectangular = new QLineEdit;
this->isRectangular->setReadOnly(true);

this->area = new QLineEdit;
this->area->setReadOnly(true);

this->perimeter = new QLineEdit;
this->perimeter->setReadOnly(true);

this->angles = new QTextEdit;
this->angles->setReadOnly(true);

this->heights = new QTextEdit;
this->heights->setReadOnly(true);

this->confirmInput = new QPushButton("Enter");
this->confirmIncrease = new QPushButton("Increase");

mainLayout->addWidget(this->side_a, 0, 0);
mainLayout->addWidget(this->side_b, 0, 1);
mainLayout->addWidget(this->side_c, 0, 2);
mainLayout->addWidget(this->confirmInput, 1, 0, 1, 3);

mainLayout->addWidget(new QLabel("Increase sides by:"), 2, 0);
mainLayout->addWidget(this->increaseSidesBy, 2, 1);
mainLayout->addWidget(this->confirmIncrease, 2, 2);

mainLayout->addWidget(new QLabel("Is rectangular:"), 3, 0);
mainLayout->addWidget(this->isRectangular, 3, 1, 1, 2);

mainLayout->addWidget(new QLabel("Area:"), 4, 0);
mainLayout->addWidget(this->area, 4, 1, 1, 2);

mainLayout->addWidget(new QLabel("Perimeter:"), 5, 0);
mainLayout->addWidget(this->perimeter, 5, 1, 1, 2);

mainLayout->addWidget(new QLabel("Angles:"), 6, 0);
mainLayout->addWidget(this->angles, 6, 1, 1, 2);

mainLayout->addWidget(new QLabel("Heights:"), 7, 0);
mainLayout->addWidget(this->heights, 7, 1, 1, 2);

connect(this->confirmInput, &QPushButton::released, this,
        &Widget::onInputConfirm);

```

```

connect(this->confirmIncrease, &QPushButton::released, this,
        &Widget::onInputIncrease);

connect(this, &Widget::valueChanged, &Widget::onValueChanged);

setLayout(mainLayout);
}

```

6 8 10

Enter

Increase sides by: Increase

Is rectangular: true

Area: 24

Perimeter: 24

Angles:

ABC: 53.1301
BCA: 36.8699
CAB: 90

Heights:

AK: 8
BK: 6
CK: 4.8

Рис.1. Работа програми (обчислення характеристик трикутника)

8 10 12

Enter

Increase sides by: 2 Increase

Is rectangular: false

Area: 39.6863

Perimeter: 30

Angles:

ABC: 55.7711
BCA: 41.4096
CAB: 82.8192

Heights:

AK: 9.92157
BK: 7.93725
CK: 6.61438

Рис.2. Робота програми (збільшення сторін трикутника)

Висновок:

У ході лабораторної роботи №5 я навчився створювати класи, використовувати різні типи доступу до полів та методів класів та конструктори для ініціалізації об'єктів, опанував принципи створення функцій-членів.