

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 № 5**

«Створення та використання 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.

$\Sigma$  = \_\_\_\_\_

**Тема роботи:** Створення та використання класів.

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

### 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. Створений 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).

## 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

### Індивідуальне завдання

1. Створити клас відповідно до варіанту (див. Додаток).
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.

### Варіант завдання

9. 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 медіан 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.

### Хід роботи

Виконання роботи розпочнемо зі створення класу трикутника, заданого завданням лабораторної роботи. Назвемо його Triangle і опишемо його наступним чином у заголовочному файлі Triangle.h:

```
#ifndef TRIANGLE_H
```

```
#define TRIANGLE_H
```

```
class TriangleViewer;
```

```
class Triangle
```

```
{
```

```
    double sideA, sideB, sideC;
```

```
    bool isExist(const double sideA, const double sideB, const double sideC);
```

```
public:
```

```
    Triangle();
```

```

Triangle(const double sideA, const double sideB, const double sideC);
Triangle(const Triangle& other);

void increaseSidesBy(const double delta);
void setSides(const double sideA, const double sideB, const double sideC);
void getSides(double &sideA, double &sideB, double &sideC) const;

double getArea() const;
void getAngles(double &angleA, double &angleB, double &angleC) const;
double getPerimeter() const;
void getMedians(double &medianA, double &medianB, double &medianC) const;
bool isRectangular() const;

void applyToForm(TriangleViewer& form);
void getFromForm(const TriangleViewer& form);
};

#endif // TRIANGLE_H

```

Як бачимо, тут я задав всі функції, які вимагало від мене завдання. Тепер напишемо реалізацію цих методів у файлі Triangle.cpp:

```

#include "triangle.h"
#include "triangleviewer.h"

bool Triangle::isExist(const double sideA, const double sideB, const double sideC)
{
    return (sideA + sideB > sideC) && (sideA + sideC > sideB) && (sideB + sideC > sideA)
    && (sideA > 0) && (sideB > 0) && (sideC > 0);
}

Triangle::Triangle() : sideA(1), sideB(1), sideC(1){}

Triangle::Triangle(const double sideA, const double sideB, const double sideC)
{
    if(isExist(sideA, sideB, sideC))
    {
        this->sideA = sideA;
        this->sideB = sideB;
        this->sideC = sideC;
    }
    else
    {

```

```

        this->sideA = this->sideB = this->sideC = 1;
    }
}

```

```

Triangle::Triangle(const Triangle &other) : sideA(other.sideA), sideB(other.sideB),
sideC(other.sideC){}

```

```

void Triangle::increaseSidesBy(const double delta)
{
    if(isExist(sideA + delta, sideB + delta, sideC + delta))
    {
        sideA += delta;
        sideB += delta;
        sideC += delta;
    }
}

```

```

void Triangle::setSides(const double sideA, const double sideB, const double sideC)
{
    if(isExist(sideA, sideB, sideC))
    {
        this->sideA = sideA;
        this->sideB = sideB;
        this->sideC = sideC;
    }
}

```

```

void Triangle::getSides(double &sideA, double &sideB, double &sideC) const
{
    sideA = this->sideA;
    sideB = this->sideB;
    sideC = this->sideC;
}

```

```

double Triangle::getArea() const
{
    const double p = (this->sideA + this->sideB + this->sideC)/2;
    return sqrt(p*(p-sideA)*(p-sideB)*(p-sideC));
}

```

```

void Triangle::getAngles(double &angleA, double &angleB, double &angleC) const
{
    angleA = acos((sideB * sideB + sideC * sideC - sideA * sideA)/(2*sideB*sideC));
    angleB = acos((sideA * sideA + sideC * sideC - sideB * sideB)/(2*sideA*sideC));
    angleC = acos((sideA * sideA + sideB * sideB - sideC * sideC)/(2*sideA*sideB));
}

```

```

}

double Triangle::getPerimeter() const
{
    return sideA + sideB + sideC;
}

void Triangle::getMedians(double &medianA, double &medianB, double &medianC) const
{
    medianA = sqrt(2*sideB*sideB + 2*sideC*sideC - sideA*sideA)/2;
    medianB = sqrt(2*sideA*sideA + 2*sideC*sideC - sideB*sideB)/2;
    medianC = sqrt(2*sideA*sideA + 2*sideB*sideB - sideC*sideC)/2;
}

bool Triangle::isRectangular() const
{
    const double eps = 1e-6;
    return (fabs(sideA*sideA + sideB*sideB - sideC*sideC) < eps ||
            fabs(sideA*sideA + sideC*sideC - sideB*sideB) < eps ||
            fabs(sideB*sideB + sideC*sideC - sideA*sideA) < eps);
}

void Triangle::applyToForm(TriangleViewer &form)
{
    form.setTriangle(*this);
}

void Triangle::getFromForm(const TriangleViewer &form)
{
    Triangle triangle = form.getTriangle();
    setSides(triangle.sideA, triangle.sideB, triangle.sideC);
}

```

Ну а тепер реалізуємо віконний додаток для демонстрації можливостей програми:

1. Реалізуємо дизайн форми, він має наступний вигляд:

New Type Here

Image

Sides

Side a:  
Side b:  
Side c:

Increase sides

Increase

☐ Show sides    ☐ Show medians  
☐ Show vertices

Information

Perimeter:  
Area:  
Angles:  
Medians:  
Is rectangular:

Set sides

Side a  
Side b  
Side c

Set sides

Рис 5.1 Дизайн форми додатку

- Створимо дизайн форми діалогу `NewTriangleDialog`, який згодом будемо використовувати для демонстрації можливості вводу/виводу на форму основного вікна:

Side a  
Side b  
Side c

OK  
Cancel

Рис 5.2 Діалог створення трикутника



3. Реалізуємо головну форму і діалог. Нижче я приведу основні фрагменти коду, які безпосередньо взаємодіють з трикутником(в файлі TriangleViewer.h та NewTriangleDialog.h об'явлено поле типу Triangle з назвою triangle) :

TriangleViewer.cpp:

```
void TriangleViewer::updateInterface()
{
    double a, b, c;
    triangle.getSides(a, b, c);
    ui->sideALabel->setText(tr("Side a: %1").arg(a));
    ui->sideBLabel->setText(tr("Side b: %1").arg(b));
    ui->sideCLabel->setText(tr("Side c: %1").arg(c));

    ui->perimeterLabel->setText(tr("Perimeter: %1").arg(triangle.getPerimeter()));
    ui->areaLabel->setText(tr("Area: %1").arg(triangle.getArea()));

    double angA, angB, angC;
    triangle.getAngles(angA, angB, angC);

    ui->anglesLabel->setText(tr("Angles: A: %1°; B: %2°; C: %3°").arg(angA *
(180/M_PI)).arg(angB * (180/M_PI)).arg(angC * (180/M_PI)));

    double medA, medB, medC;
    triangle.getMedians(medA, medB, medC);
    ui->mediansLabel->setText(tr("Medians: Ma: %1; Mb: %2; Mc:
%3").arg(medA).arg(medB).arg(medC));

    ui->rectangularLabel->setText(tr("Is rectangular: %1").arg(triangle.isRectangular() ?
"true" : "false"));

    ui->viewWidget->setTriangle(this->triangle);
    ui->viewWidget->update();
}

void TriangleViewer::on_increaseButton_clicked()
{
    double delta = ui->increaseField->text().toDouble();
    ui->increaseField->setText("");

    triangle.increaseSidesBy(delta);
    updateInterface();
}

void TriangleViewer::on_setSidesButton_clicked()
```

```

{
    double a = ui->sideAField->text().toDouble();
    double b = ui->sideBField->text().toDouble();
    double c = ui->sideCField->text().toDouble();

    ui->sideAField->setText("");
    ui->sideBField->setText("");
    ui->sideCField->setText("");

    triangle.setSides(a,b,c);
    updateInterface();
}

void TriangleViewer::on_actionTriangle_triggered()
{
    NewTriangleDialog dialog(this);

    dialog.exec();
}

```

NewTriangleDialog.cpp:

```

#include "newtriangledialog.h"
#include "ui_newtriangledialog.h"
#include "triangleviewer.h"

```

```

NewTriangleDialog::NewTriangleDialog(TriangleViewer* viewer, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::NewTriangleDialog)
{
    ui->setupUi(this);
    mainWindow = viewer;

    Triangle triangle;
    triangle.getFromForm(*mainWindow);
    double a, b, c;
    triangle.getSides(a,b,c);

    ui->sideASet->setText(QString::number(a));
    ui->sideBSet->setText(QString::number(b));
    ui->sideCSet->setText(QString::number(c));
}

```

```

NewTriangleDialog::~NewTriangleDialog()

```

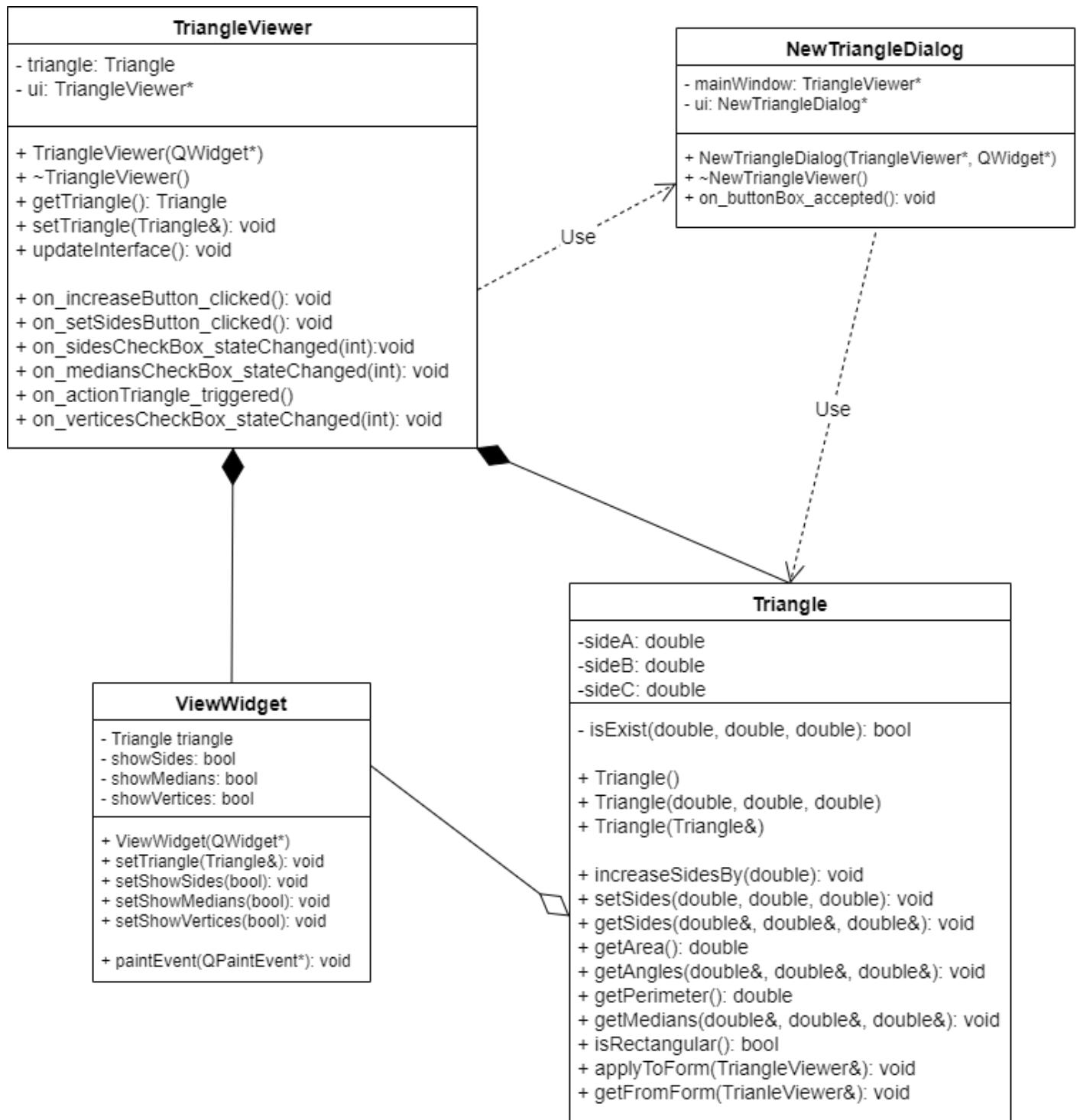
```
{
    delete ui;
}

void NewTriangleDialog::on_buttonBox_accepted()
{
    Triangle triangle
        (
            ui->sideASet->text().toDouble(),
            ui->sideBSet->text().toDouble(),
            ui->sideCSet->text().toDouble()
        );

    triangle.applyToForm(*mainWindow);
}
```

Також, програма оперує створеним мною віджетом ViewWidget, який малює переданий йому трикутник.

**UML – діаграма**

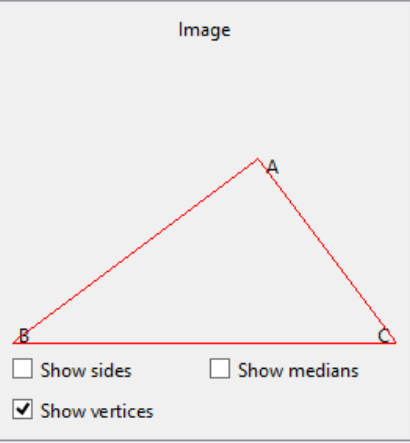


**Результат виконання:**

TriangleViewer

New

Image



☐ Show sides☐ Show medians☒ Show vertices

Sides

Side a: 10  
Side b: 6  
Side c: 8

Increase sides

Set sides

Side a  
  
Side b  
  
Side c

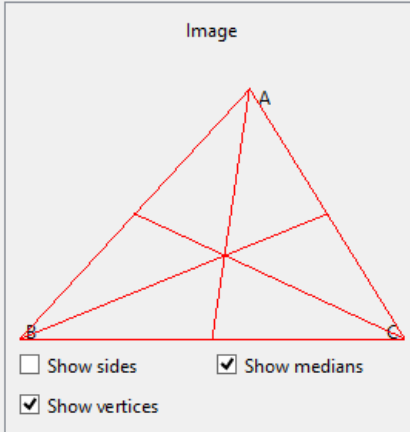
Information

Perimeter: 24  
Area: 24  
Angles: A: 90°; B: 36.8699°; C: 53.1301°  
Medians: Ma: 5; Mb: 8.544; Mc: 7.2111  
Is rectangular: true

TriangleViewer

New

Image



☐ Show sides☒ Show medians☒ Show vertices

Sides

Side a: 17  
Side b: 13  
Side c: 15

Increase sides

Set sides

Side a  
  
Side b  
  
Side c

Information

Perimeter: 45  
Area: 93.8999  
Angles: A: 74.3815°; B: 47.4315°; C: 58.187°  
Medians: Ma: 11.1692; Mb: 14.6544; Mc: 13.1434  
Is rectangular: false

#### Рис 5.3-4 Результат виконання програми

**Висновок:** виконавши лабораторну роботу №5, я навчився створювати класи, використовувати конструктори для ініціалізації об'єктів, опанував принципи створення функцій-членів. Навчився використовувати різні типи доступу до полів та методів класів. На основі отриманих знань я створив додаток для роботи з трикутником. На мою думку, робота виявилась досить продуктивною.