Об’єктно орієнтоване програмування

Лабораторна робота №11

Наслідування та поліморфізм (2)

Теоретичні відомості

Наслідування та поліморфізм. Наслідування класів передбачає збереження полів і методів (а також властивостей і подій) базового класу в класі нащадку.

Список базових класів приводиться в описі класу після імені класу і двокрапки. Елементи цього списку розділяються комами і крім імен класів містять класифікатори доступу. наприклад:

```
class D: public A, protected B, private C {...};
```

Класифікатор доступу перед ім'ям класу призначений для модифікації доступу до полів і методів класу. Модифікує класифікатор доступу вказує на максимально можливий доступ. Так все елементи класу A, з наведеного вище прикладу, при наслідуванні збережуть свої визначення доступу. Для класу B, елементи, описані в розділі public, будуть успадковані як елементи типу protected. І, нарешті, всі елементи класу C, будуть успадковані як елементи типу private. Таке наслідування використовується досить рідко, так як в цьому випадку немає можливості отримати доступ до полів і методів батьківського класу. Необхідно також мати на увазі, що модифікує класифікатором доступу за замовчуванням для класів, описаних за допомогою ключового слова class, є private. Тобто заголовок класу D міг бути записаний так:

```
class D: public A, protected B, C {...}
```

Модифікує класифікатором доступу за замовчуванням для класів, описаних за допомогою ключового слова struct, є public.

Доступ до окремих елементів базових класів, обмежений за допомогою модифікуючого класифікатора, може бути відновлений. Це проводиться шляхом явного їх опису в відповідному розділі дочірнього класу.

Наприклад, якщо в класі C в розділі public був описаний метод Mesh, то доступ до нього може бути відновлений за допомогою наступної коду в описі класу D:

```
class D: public A, protected B, C {...
public:
C :: Mesh ();
...
```

При множині спадкування екземпляр класу нащадка містить в собі як підоб'єкти екземпляри всіх базових класів. Наприклад, екземпляр класу D, буде містити екземпляри класів A, B і C (рис. 1).

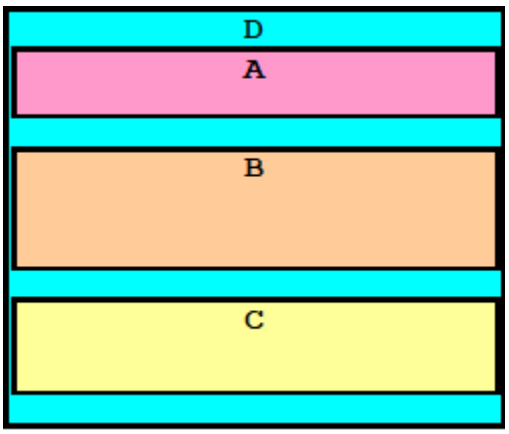

Рисунок 11.1 – Схема множинного наслідування

Синтаксисом мови C ++ заборонено явно вказувати один і той же клас в якості базового більше одного разу. Однак якщо кілька базових класів мають загального батька, то екземпляр цього класу буде включений в екземпляр класу нащадка кілька разів.

Спадкування тісно пов'язане з поняттям поліморфізму. Під поліморфізмом розуміють відмінність в реалізації однойменних методів у родинних класах. Механізмом забезпечує поліморфну поведінку є динамічне зв'язування.

При статичному зв'язуванні, яке проводиться для звичайних методів, реалізація методу визначається на етапі компіляції і залежить від типу змінної, що викликала метод.

При динамічному зв'язуванні, яке проводиться для віртуальних методів, реалізація методу визначається під час роботи програми і залежить від конкретного класу, на екземпляр якого посилається в даний момент змінна, що викликає метод.

Віртуальний метод (віртуальна функція) - метод класу, який може бути перевизначений в класах-спадкоємців так, що конкретна реалізація методу для виклику буде визначатися під час виконання. Таким чином, програмісту необов'язково знати точний тип об'єкта для роботи з ним через віртуальні методи: досить лише знати, що об'єкт належить класу або спадкоємцю класу, в якому метод оголошений.

Для оголошення віртуальних методів використовується директива virtual, яка записується перед заголовком методу в описі класу. наприклад:

```
class Base
{
public:
void SM () {cout << "Base SM \ n";};
virtual void VM () {cout << "Base VM \ n";};
};
```

Віртуальні методи можуть не мати реалізації. Такі методи називають абстрактними (pure virtual methods). Для того щоб описати такий метод, після його заголовка ставлять = 0.

наприклад:

```
virtual void Go () = 0;
```

Клас який містить явно описаний чи успадкований абстрактний метод називається абстрактним. Абстрактні класи можуть бути використані в якості базових, з подальшою реалізацією всіх абстрактних методів.

Приклад. Написати програму, яка дозволяє користувачеві виставляти необмежену кількість пішок на порожню шахову дошку і визначає по кліку на фігуру клітини шахової дошки, на які ця фігура може зробити хід (рис. 11.2)

Лістинг 11.1 – Файл chesspiece.h

```
#ifndef CHESSPIECE_H
#define CHESSPIECE_H
#include <QString>
#include <QPainter>
const int DeskBorder = 15;
const int DeskCell = 50;

class ChessPiece
{
protected:
    int fx, fy;
    QString imagename;
public:
    ChessPiece(int fx, int fy);
    virtual ~ChessPiece();
    void setPosition(int x, int y);
    QPoint getPosition();
    void Draw(QPainter *p);
    void DrawShadow(QPainter *p, int i, int j);
    virtual bool canMove(int x, int y) = 0;
    static QRect GetCellRect(int i, int j);
    static QPoint GetCellByMouseClick(QPoint p);
};
#endif // CHESSPIECE_H
```

Лістинг 11.1 – Файл chesspiece.cpp

```
#include "chesspiece.h"
#include <qdebug.h>
#include <qdir.h>
ChessPiece::ChessPiece(int fx, int fy)
{
    imagename =  QDir::currentPath() + "/" + imagename;
    setPosition(fx, fy);
}
ChessPiece::~ChessPiece()
{
}
QRect ChessPiece::GetCellRect(int i, int j)
{
    QRect result;
    int nj = 7 - j;
    result.setLeft(i * DeskCell + DeskBorder);
```

```
    result.setTop(nj * DeskCell + DeskBorder);
    result.setRight((i + 1)*DeskCell + DeskBorder);
    result.setBottom((nj + 1)*DeskCell + DeskBorder);
    return result;
}
QPoint ChessPiece::getPosition()
{
    return QPoint(fx, fy);
}
QPoint ChessPiece::GetCellByMouseClick(QPoint p)
{
    int x = p.x();
    int y = p.y();
    x -= DeskBorder;
    y -= DeskBorder;
    if(x < 0 || y < 0 || x > 8 * DeskCell || y > 8 * DeskCell)
        return QPoint(-1, -1);
    return QPoint(x / DeskCell, 7 - y / DeskCell);
}
void ChessPiece::Draw(QPainter *p)
{
    QImage img(imagename);
    p->drawImage(GetCellRect(fx, fy), img, img.rect());
}
void ChessPiece::DrawShadow(QPainter *p, int i, int j)
{

    QImage img(imagename);
    p->setCompositionMode(QPainter::CompositionMode_Overlay);
    p->drawImage(GetCellRect(i, j), img, img.rect());
    p->setCompositionMode(QPainter::CompositionMode_SourceOver);
}
void ChessPiece::setPosition(int x, int y)
{
    fx = x;
    fy = y;
}
```

Лістинг 11.1 – Файл whitepawn.h

```
#ifndef WHITEPAWN_H
#define WHITEPAWN_H
#include "chesspiece.h"

class WhitePawn : public ChessPiece
{
public:
    WhitePawn(int fx, int fy);
    bool canMove(int x, int y);
};
#endif // WHITEPAWN_H
```

Лістинг 11.1 – Файл whitepawn.cpp

```
#include "whitepawn.h"
WhitePawn::WhitePawn(int fx, int fy): ChessPiece(fx, fy)
{
    imagename = "./chess_images/Chess_plt60.png";
}
bool WhitePawn::canMove(int x, int y)
{
    if(x != fx)
        return false;
    if(fy == 1)
        if(y == 3 || y == 2)
            return true;
        else
            return false;
    else if(y == fy + 1)
        return true;
    else
        return false;
}
```

Лістинг 11.1 – Файл blackpawn.h

```
#ifndef BLACKPAWN_H
#define BLACKPAWN_H
#include "chesspiece.h"

class BlackPawn : public ChessPiece
{
public:
    BlackPawn(int fx, int fy);
    bool canMove(int x, int y);
};

#endif // BLACKPAWN_H
```

Лістинг 11.1 – Файл blackpawn.cpp

```
#include "blackpawn.h"

BlackPawn::BlackPawn(int fx, int fy): ChessPiece(fx, fy)
{
    imagename = "./chess_images/Chess_pdt60.png";
}

bool BlackPawn::canMove(int x, int y)
{
    if(x != fx)
        return false;
    if(fy == 6)
        if(y == 5 || y == 4)
            return true;
        else
            return false;
    else if(y == fy - 1)
```

```
        return true;
    else return false;
}
```

Лістинг 11.1 – Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "chesspiece.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    void DrawChessField(QPainter *p);
    bool eventFilter(QObject* watched, QEvent* event);
    bool StrToChessPos(QString s,  int &i,int &j);
    QVector<ChessPiece*> pieces;
    QPoint cellcoord;
};
#endif // MAINWINDOW_H
```

Лістинг 11.1 – Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPainter>
#include <QDebug>
#include <QKeyEvent>
#include "chesspiece.h"
#include "whitepawn.h"
#include "blackpawn.h"
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
```

```
    ui->widget->installEventFilter(this);
}

MainWindow::~MainWindow()
{
    delete ui;
    for(auto piece = pieces.begin(); piece < pieces.end();
piece++)
        delete *piece;
}

bool MainWindow::eventFilter(QObject* watched, QEvent* event)
{
    if(watched == ui->widget && event->type() == QEvent::Paint)
    {
        QPainter p(ui->widget);
        DrawChessField(&p);
        for(auto piece = pieces.begin(); piece < pieces.end();
piece++)
        {
            (*piece)->Draw(&p);
            if((*piece)->getPosition() ==  this->cellcoord)
                for(int i = 0; i < 8; i++)
                    for(int j = 0; j < 8; j++)
                        if((*piece)->canMove(i, j))
                            (*piece)->DrawShadow(&p, i, j);
        }

        return true;
    }

    if(watched == ui->widget && event->type() ==
QEvent::MouseButtonPress)
    {
        QPoint clickpos = reinterpret_cast<QMouseEvent
*>(event)->pos();
        QPoint coord =
ChessPiece::GetCellByMouseClick(clickpos);
        this->cellcoord = coord;
        ui->widget->update();
    }

    return false;
}
void MainWindow::DrawChessField(QPainter *p)
{
    QBrush b1(QColor(0xEEEE00));
    QBrush b2(QColor(0x666600));

    for(int i = 0; i < 8; i++)
        for(int j = 0; j < 8; j++)
        {
            if((i + j) % 2 == 0)
```

```
                p->fillRect(ChessPiece::GetCellRect(i, j), b1);
            else
                p->fillRect(ChessPiece::GetCellRect(i, j), b2);

        }
    for(int i = 0; i < 8; i++)
    {
        QChar text = QChar('A' + i);

        QRect textrect1 = p-
>fontMetrics().boundingRect(QString(text));
        QRect textrect2 = p-
>fontMetrics().boundingRect(QString::number(8 - i));

        p->drawText(DeskBorder + i * DeskCell + DeskCell / 2 -
textrect1.width() / 2, 2 * DeskBorder + 8 * DeskCell,
QString(text));
        p->drawText(DeskBorder / 2, DeskBorder + i * DeskCell +
DeskCell / 2 + textrect2.height() / 2, QString::number(8 - i));
    }
}



bool MainWindow::StrToChessPos(QString s,  int &i, int &j)
{
    if(s.length() != 2)
        return false;
    QByteArray sc = s.toUpper().toLocal8Bit();
    i = sc[0] - 'A';
    j = sc[1] - '1';
    return true;
}

void MainWindow::on_pushButton_clicked()
{
    int x = 0, y = 0;
    if(!StrToChessPos(ui->lineEdit->text(), x, y))
        return;
    for(auto piece = pieces.begin(); piece < pieces.end();
piece++)
    {
        if((*piece)->getPosition() == QPoint(x, y))
        {
            QMessageBox::warning(this, "", "Клітинка зайнята");
            return;
        }
    }


    if(ui->radioButton->isChecked())
        pieces.push_back(new BlackPawn(x, y));
```

```
    if(ui->radioButton_2->isChecked())
        pieces.push_back(new WhitePawn(x, y));
    ui->widget->update();
}
```

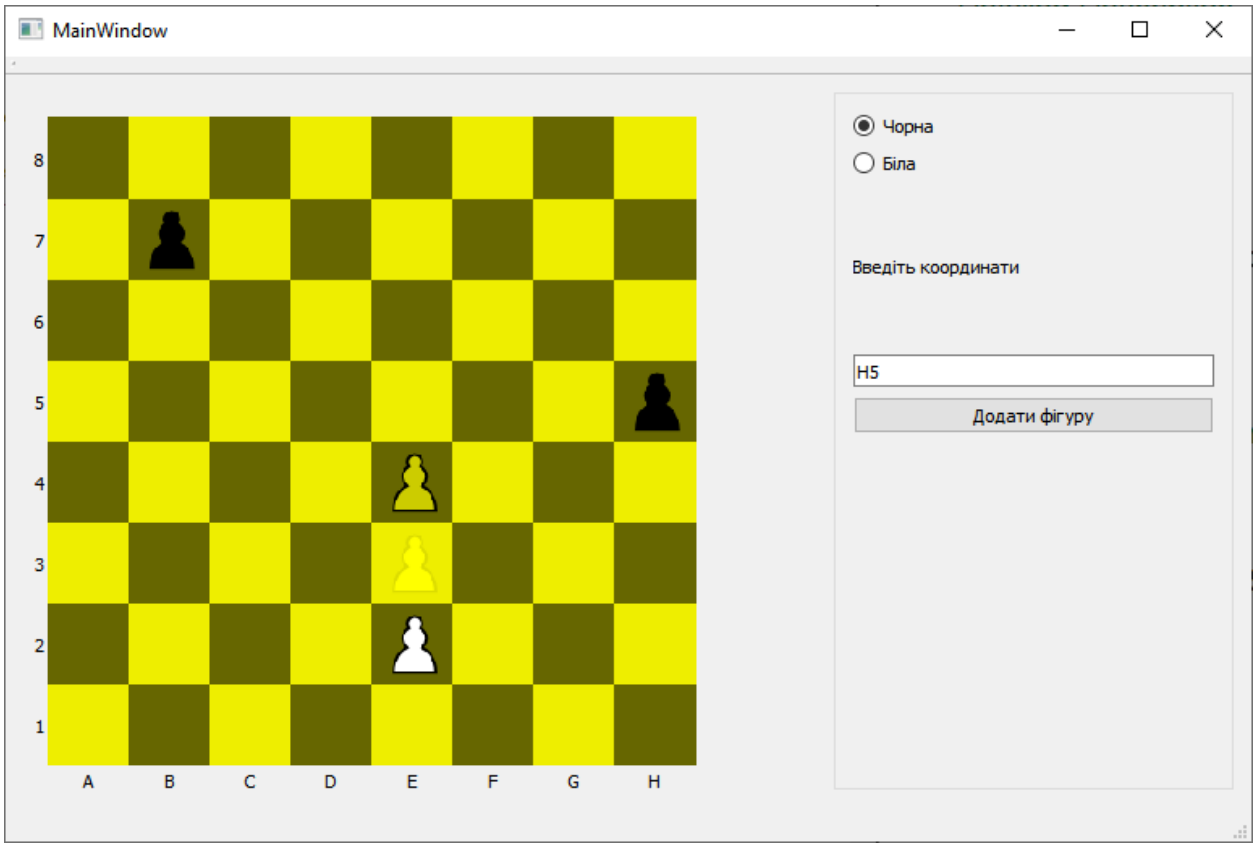

Рисунок 11.2 – Результат роботи програми

Варіанти завдань

УВАГА, зображення фігур доступні за посиланням https://github.com/a-vodka/oop_qt/examples/lab11-chess/chess_images/ . За бажанням можна використовувати інші зображення.

Варіант 1.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і ферзів (теж по одному) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 2.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і тур (теж по одній) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 3.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і слонів (по 2 кожного кольору) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 4.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і коней (по 2 кожного кольору) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 5.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і пішаків (по 4 кожного

кольору) і виконувати ходи цими фігурами. При досягнені пішаками краю поля, вони повинні перетворюватись на ферзів. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 6.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і тур (по 2 кожного кольору) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 7.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і слонів (по 3 кожного кольору) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 8.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і коней (по 3 кожного кольору) і виконувати ходи цими фігурами. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.

Варіант 9.

Написати програму, яка дозволяє двом користувачам виставляти на шахівниці королів (по одному кожного кольору) і пішаків (по 5 кожного кольору) і виконувати ходи цими фігурами. . При досягнені пішаками краю поля, вони повинні перетворюватись на ферзів. Програма повинна контролювати правильність ходів фігур, підсвічувати можливі ходи для обраної фігури і генерувати повідомлення про помилки в разі виконання некоректних операцій. Уразі загибелі короля, виводиться повідомлення про закінчення гри.