**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

**Інститут комп’ютерних наук та інформаційних технологій**

***Кафедра ПЗ***

**Звіт**

до лабораторної роботи №9

на тему «Принцип поліморфізму»

з курсу «Об’єктно-орієнтоване програмування»

**Виконала:**

студентка групи ПЗ-11

Дерпак Вікторія

**Перевірила**:

доц. Коротєєва Т.О.

Львів

**2022**

**Тема.**Наслідування. Створення та використання ієрархії класів.

**Мета.** Навчитись створювати списки об’єктів базового типу, що включають об’єкти похідних типів. Освоїти способи вирішення проблеми неоднозначності при множинному наслідуванні. Вивчити плюси заміщення функцій при множинному наслідуванні. Навчитись використовувати чисті віртуальні функції, знати коли варто використовувати абстрактні класи.

**Теоретичні відомості:**

Поліморфізм – одна з трьох основних парадигм ООП. Якщо говорити коротко, поліморфізм – це здатність об'єкта використовувати методи похідного класу, який не існує на момент створення базового.

В ооп часто створюються ієрархії логічно пов’язаних класів. Наприклад уявимо клас Shape, від якого наслідуються класи Rectangle і Circle. Пізніше від класу Rectangle наслідується клас Square, як окремий вид прямокутників. В класі Shape немає змісту визначати функції знаходження площі чи знаходження периметру, оскільки в похідних класах вони будуть перевизначені. Всі методи повинні функціонувати нормально в похідних класах а не в базовому класі Shape, оскільки неможливо створити екземпляр форми як такий. Також програма повинна бути захищена від спроби користувача створити об’єкт цього класу.

С++ підтримує створення абстрактних типів даних з чистими віртуальними функціями. Чисті віртуальні функції це такі, які ініціалізуються нульовим значенням, наприклад

virtual void Draw() = 0;

Клас, який містить чисті віртуальні функції є абстрактним. Неможливо створити об’єкт абстрактного класу. Поміщення в клас чистої віртуальної функції означає наступне:

* неможливо створити об’єкт цього класу;
* необхідно замістити чисту віртуальну функцію в похідному класі.

Будь-який клас, наслідуваний від абстрактного класу, наслідує від нього чисту віртуальну функцію, яку необхідно замістити щоб отримати можливість створювати об’єкти цього класу.

Зазвичай чисті віртуальні функції оголошуються в абстрактному базовому класі і не виконуються. Оскільки неможливо створити об’єкт абстрактного базового класу, як правило, нема необхідності і в виконання чистої віртуальної функції.

Тим не менше, інколи виникає необхідність виконання чистої віртуальної функції. Вона може бути викликана з об’єкта, який наслідує абстрактний клас, наприклад, щоб забезпечити загальну функціональність для всіх заміщених функцій

**Завдання для лабораторної роботи:**

1. Розробити ієрархію класів відповідно до варіанту.
2. Використати множинне наслідування, продемонструвати вирішення проблеми з неоднозначністю доступу до членів базових класів за допомогою віртуального наслідування, за допомогою явного звертання до членів класу та за допомогою заміщення функцій в похідному класі (при потребі).
3. Створити списки об’єктів базового типу, в них помістити об’єкти похідного типу. Продемонструвати виклик функцій з об’єктів – елементів списку. Використати опратор dynamic\_cast (при потребі).
4. Створити абстрактний клас, використати чисто віртуальну функцію, що містить реалізацію в базовому класі.
5. Для вивільнення динамічної пам’яті використовувати віртуальні деструктори.
6. Сформувати звіт до лабораторної роботи. Відобразити в ньому діаграму наслідування класів.

Варіант 8.

Абонемент в АкваПарк

Базовий абстрактний клас WPSubscription. Далі SingleWPSubscription, GroupWPSubscription, FamilyWPSubscription, можливо, CorporateWPSubscription. Для кожного з цих класів можна додати окремі властивості, як то доступ до окремих зон, додаткові послуги, а також змінювати методи визначення вартості, відповідно до підписки. Базовий же клас має визначати, до прикладу, тривалість по часу, загальну вартість.

**Результати:**

“WPSubscription.h”

#ifndef WPSUBSCRIPTION\_H

#define WPSUBSCRIPTION\_H

*class* **WPSubscription**

{

*protected*:

int price;

int time;

*public*:

*virtual* void ***CalculatePrice***()=0;

*virtual* ~***WPSubscription***(){}

int **GetPrice**(){*return* price;}

};

#endif *//* *WPSUBSCRIPTION\_H*

“SingleWPSubscription.h”

#ifndef SINGLEWPSUBSCRIOTION\_H

#define SINGLEWPSUBSCRIOTION\_H

#include "WPSubscription.h"

*class* **SingleWPSubscription**: *public* *virtual* WPSubscription

{

*protected*:

bool hasTrainer;

bool hasLunch;

*public*:

**SingleWPSubscription**(bool hastrainer, bool haslunch, int time)

{

*this*->time=time;

price=0;

hasTrainer=hastrainer;

hasLunch=haslunch;

}

*virtual* ~***SingleWPSubscription***(){}

*virtual* void ***CalculatePrice***()

{

price = time\*150;

*if*(hasTrainer)

price+=100\*time;

*if*(hasLunch)

price+=70\*time;

}

};

#endif *//* *SINGLEWPSUBSCRIOTION\_H*

“GroupWPSubscription.h”

#ifndef GROUPWPSUBSCRIPTION\_H

#define GROUPWPSUBSCRIPTION\_H

#include "WPSubscription.h"

*class* **GroupWPSubscription**: *public* *virtual* WPSubscription

{

*protected*:

int peopleNumber;

bool hasChildrenUnderTen;

*public*:

**GroupWPSubscription**(int peoplenumber, bool haschund10, int time)

{

price=0;

*this*->time=time;

peopleNumber=peoplenumber;

hasChildrenUnderTen=haschund10;

}

*virtual* ~***GroupWPSubscription***(){}

*virtual* void ***CalculatePrice***()

{

price = time\*100\*peopleNumber;

*if*(hasChildrenUnderTen)

price-=50\*peopleNumber;

}

};

#endif *//* *GROUPWPSUBSCRIPTION\_H*

“FamilyWPSubscription.h”

#ifndef FAMILYWPSUBSCRIPTION\_H

#define FAMILYWPSUBSCRIPTION\_H

#include "GroupWPSubscription.h"

#include "SingleWPSubscriotion.h"

*class* **FamilyWPSubscription**: *public* GroupWPSubscription, *public* SingleWPSubscription

{

*public*:

**FamilyWPSubscription**(int peoplenumber, bool hastrainer, bool haslunch, bool haschund10, int time) : GroupWPSubscription(peoplenumber, haschund10, time), SingleWPSubscription(hastrainer, haslunch, time)

{

}

*virtual* ~***FamilyWPSubscription***(){}

void ***CalculatePrice***() *override*

{

price=peopleNumber\*time\*120;

*if*(hasLunch)

price+=peopleNumber\*time\*40;

*if*(hasTrainer)

price+=peopleNumber\*time\*100;

*if*(hasChildrenUnderTen)

price-=50\*peopleNumber;

}

};

#endif *//* *FAMILYWPSUBSCRIPTION\_H*

“mainwindow.h”

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*private* slots:

void **on\_actionSingle\_triggered**();

void **on\_actionGroup\_triggered**();

void **on\_actionFamily\_triggered**();

void **on\_Calculate\_clicked**();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

“mainwindow.cpp”

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "FamilyWPSubscription.h"

#include "GroupWPSubscription.h"

#include "SingleWPSubscriotion.h"

#include <QIntValidator>

WPSubscription\* subscription = *nullptr*;

int choice=1;

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

ui->actionSingle->trigger();

QValidator \*validator = *new* QIntValidator(1, 100, *this*);

ui->Days->setValidator(validator);

validator = *new* QIntValidator(1, 9, *this*);

ui->NumberOfPeople->setValidator(validator);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

void MainWindow::**on\_actionSingle\_triggered**()

{

ui->Children->hide();

ui->Lunch->show();

ui->Trainer->show();

ui->label\_3->hide();

ui->NumberOfPeople->hide();

choice=1;

}

void MainWindow::**on\_actionGroup\_triggered**()

{

ui->Children->show();

ui->Lunch->hide();

ui->Trainer->hide();

ui->label\_3->show();

ui->NumberOfPeople->show();

choice=2;

}

void MainWindow::**on\_actionFamily\_triggered**()

{

ui->Children->show();

ui->Lunch->show();

ui->Trainer->show();

ui->label\_3->show();

ui->NumberOfPeople->show();

choice=3;

}

void MainWindow::**on\_Calculate\_clicked**()

{

*if*(choice==1)

subscription=*new* SingleWPSubscription(ui->Trainer->isChecked(), ui->Lunch->isChecked(), ui->Days->text().toInt());

*if*(choice==2)

subscription=*new* GroupWPSubscription(ui->NumberOfPeople->text().toInt(), ui->Trainer->isChecked(), ui->Days->text().toInt());

*if*(choice==3)

subscription=*new* FamilyWPSubscription(ui->NumberOfPeople->text().toInt(), ui->Trainer->isChecked(), ui->Lunch->isChecked(), ui->Children->isChecked(), ui->Days->text().toInt());

subscription->*CalculatePrice*();

ui->Price->setText(QString::number(subscription->GetPrice()));

*delete* subscription;

}

“main.cpp”

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

*return* a.exec();

}

“mainwindow.ui”

Зображення, що містить текст, знімок екрана, монітор, чорний

Автоматично згенерований опис

Результати виконання програми:

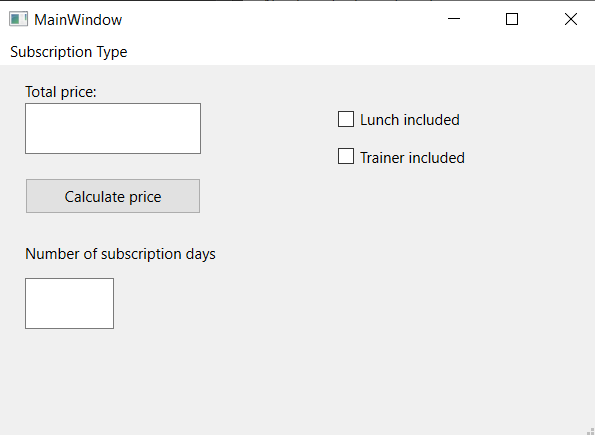


Рис. . Запуск програми

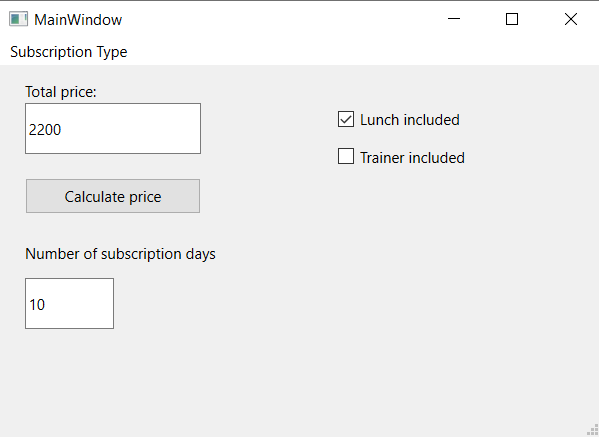


Рис. . Виконання програми з типом підписки "Single"

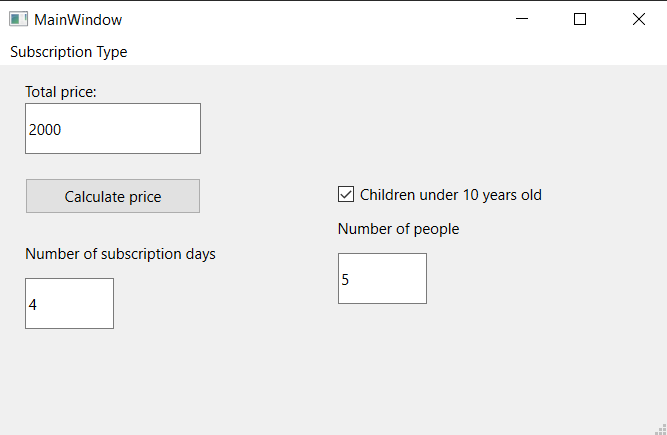


Рис. . Виконання програми з типом підписки "Group"

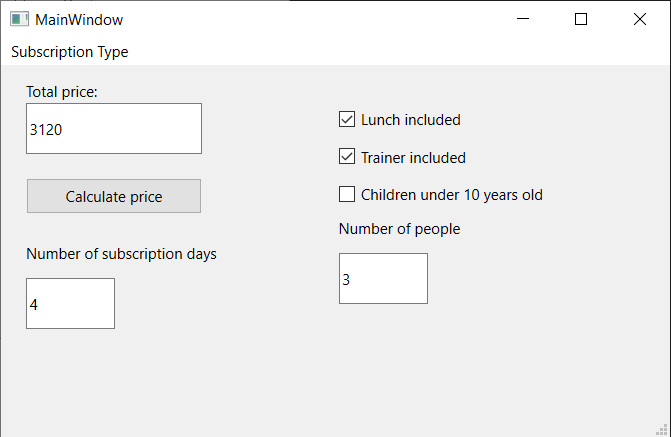


Рис. . Виконання програми з типом підписки "Family"

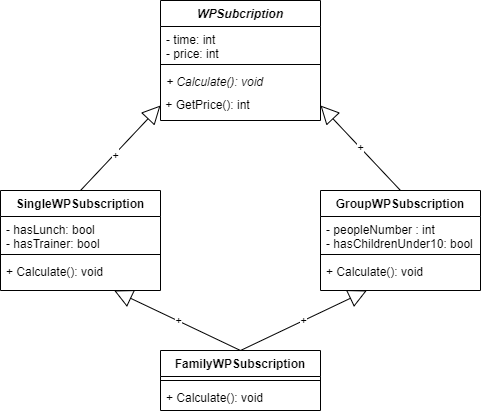


Рис. . UML-діаграма класів

**Висновок:** Виконуючи лабораторну роботу №9 я навчилась створювати списки об’єктів базового типу, що включають об’єкти похідних типів. Освоїла способи вирішення проблеми неоднозначності при множинному наслідуванні. Вивчила плюси заміщення функцій при множинному наслідуванні. Навчилась використовувати чисті віртуальні функції, знати коли варто використовувати абстрактні класи.