МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ „ЛЬВІВСЬКА ПОЛІТЕХНІКА”

**КЛАСИ, ІНТЕРФЕЙСИ І СТРУКТУРИ**

**У МОВІ ПРОГРАМУВАННЯ С#**

**МЕТОДИЧНІ ВКАЗІВКИ**

до практичних занять

з дисципліни „Конструювання програмного забезпечення”

для студентів базового напряму

„Програмна інженерія”

|  | *Затверджено*  *на засіданні кафедри*  *програмного забезпечення*  *Протокол № 14 від 18.04.2013 р.* |
| --- | --- |

Львів – 2017

**Тема.:** Методичні вказівки до практичних занять з дисципліни „Конструювання програмного забезпечення” для студентів базового напряму „Програмна інженерія” / Укл.: П. В. Сердюк, О.О. Сівець. – Львів: Видавництво Національного університету „Львівська політехніка”, 2013. – 28 с.

**Укладачі**

Сердюк. П. В., к. т. н., доц. кафедри програмного забезпеченння національного університету “Львівська політехніка”

Сівець О. О., асист. кафедри програмного забезпечення національного університету “Львівська політехніка”

**Відповідальний за випуск**

Федасюк Д. В., д-р тех. наук, проф., завідувач кафедрою програмного забезпечення, проректор з науково-педагогічної роботи національного університету “Львівська політехніка”

**Рецензенти**

Гавриш В. І., к.ф.-м.н., доц. кафедри програмного забезпеченння національного університету “Львівська політехніка”

Огородник Н. П., к.ф.-м.н., асис. кафедри теорії оптимальних процесів Львівського національного університету ім. І. Франка

Зміст

[1.](#_heading=h.lnxbz9) Вступ 4

[2.](#_heading=h.1fob9te) Поняття інтерфейсів 4

[Використання інтерфейсів 5](#_heading=h.3znysh7)

[Оголошення інтерфейсу 5](#_heading=h.2et92p0)

[Реалізація інтерфейсу 6](#_heading=h.tyjcwt)

[Комбіновані інтерфейси 7](#_heading=h.3dy6vkm)

[Інтерфейси і наслідування класів 11](#_heading=h.1t3h5sf)

[Властивості в інтерфейсах 14](#_heading=h.4d34og8)

[Індексатори в інтерфейсах 15](#_heading=h.2s8eyo1)

[3.](#_heading=h.17dp8vu) Хід роботи 18

[4.](#_heading=h.3rdcrjn) Варіанти завдань 21

[5.](#_heading=h.26in1rg) Контрольні запитання. 24

**ПЛАТФОРМА .NET**

**Мета роботи:** Ознайомлення з основами класів, структур, та інших базових елементів мови програмування С#.

Після виконання лабораторної студент повинен уміти створювати класи, інтерфейси та структури .

**Теоретичні відомості.**

# **Вступ**

Поняття класів, структур є базовими для мов програмування ООП, таких як С++ та інші. С# як і мова Java, вводить ще одне нове поняття – інтерфейс, який дозволяє розділяти бізнес логіку і є значним посиленням механізму інкапсуляції у С#. Тому у даних методичних матеріалах ми більше сконцентруємось на понятті інтерфейсу та його використанню.

# **Поняття інтерфейсів**

Клас С# може успадковувати властивості лише одного базового класу. Таким чином, на відміну від інших мов ООП (наприклад, C++) у мові С# кожен похідний клас може мати лише один базовий клас.

На перший погляд таке обмеження може здатися досить істотним, але у багатьох випадках воно успішно обходиться за допомогою механізмів інтерфейсів.

Тоді як класи є механізмом для представлення деяких об’єктів (таких, наприклад, як геометричні фігури, телевізори і т. п.), інтерфейси застосовуються для опису дій над цими об’єктами.

Поняття інтерфейсу очевидне на звичайному життєвому рівні. Уявіть собі наприклад, плеєр для програвання компакт-дисків. Існує незліченна кількість різних моделей таких плеєрів, що відрізняються один від одного формою корпусу, кольором, розміром і іншими атрибутами. Проте всі вони мають практично однаковий набір кнопок, за допомогою яких можна запускати або зупиняти програвання компакт-диска, переходити з однієї доріжки на іншу, а також витягувати компакт-диск з корпусу плеєра.

Така уніфікація «користувацького інтерфейсу» плеєра дозволяє будь-кому з вас швидко освоїти новий плеєр, не знаючись на деталях його внутрішньої будови. Аналогічно ви легко зумієте скористатися банкоматом будь-якої моделі для зняття грошей з кредитної картки, оскільки всі банкомати мають один і той же «інтерфейс».

Повертаючись до мови С#, відмітимо, що використання різноманітних властивостей, індексаторів, подій, а також інтерфейсів дозволяє створювати об'єкти, які можна використовувати, не вникаючи в деталі їх реалізації. При цьому об'єкт (клас) може реалізувати набір інтерфейсів, кожен з яких відповідає за виконання над об'єктом певних дій.

Інтерфейси найбільш схожі на віртуальні методи абстрактного класу, які мають бути визначені в базовому класі. Хорошою новиною є те, що кожен клас С# може реалізувати довільну кількість інтерфейсів. Саме ця обставина робить неістотним обмеження С#, що стосується можливості множинного наслідування класів.

## **Використання інтерфейсів**

Інтерфейси оголошуються за допомогою ключового слова interface, аналогічно класам. Всередині оголошення інтерфейсу необхідно перерахувати методи, з яких складається інтерфейс. Окрім методів, усередині інтерфейсів можна також оголошувати властивості, індексатори і події.

Клас, що реалізує інтерфейс, повинен містити в собі тіло методів, оголошених в рамках всіх інтерфейсів, що реалізуються ним.

Щоб було зрозуміліше, наведемо конкретний приклад.

## **Оголошення інтерфейсу**

Нехай нам потрібно створити клас *Point*, що відображає поведінку крапки на площині. Цей об'єкт повинен зберігати поточні координати точки, роблячи їх доступними при допомозі властивостей X і Y. Крім того, клас *Point* повинен реалізовувати два інтерфейси *IPrint і IMail*, перший з яких дозволяє роздруковувати крапку на принтері і виконувати попередній перегляд результатів друку, а другий — відправляти крапку по електронній пошті за заданою адресою.

Нижче приведемо оголошення інтерфейсів *IPrint і IMail*:

interface IPrint

{

void Print();

void PrintPreview();

}

interface IMail

{

void SendMail(string mailAddress);

}

Як бачите, в оголошенні інтерфейсів ми вказуємо лише прототипи методів не включаючи тіло цих методів. За конкретну реалізацію методів інтерфейсу відповідає клас, який реалізує цей інтерфейс.

Крім цього, в прототипах методів не потрібно (і не допускається) вказувати модифікатори доступу, такі, як *public* або *private*. Всі методи інтерфейсу є загальнодоступними за замовченням.

Зверніть також увагу на назви інтерфейсів. У С# прийнято, що назва інтерфейсу починається з прописної букви I. І хоча це не строге правило, рекомендується його дотримуватися, оскільки при цьому вихідний текст програми буде зрозуміліший.

## **Реалізація інтерфейсу**

При оголошенні інтерфейсів ми складаємо погодження (контракт), якому повинні задовольняти методи, властивості, індексатори і події, оголошені в рамках інтерфейсу. Що ж до конкретної реалізації інтерфейсу, то вона, як ми вже говорили, покладається на клас, що реалізовує інтерфейс.

Нижче приводимо в скороченому вигляді вихідний текст класу *Point*, що реалізує інтерфейси *IPrint* та *IMail*:

class Point : IPrint, IMail

{

protected int xPos;

protected int yPos;

public Point(int x, int у)

{

xPos = x;

yPos = у;

}

void IPrint.Print()

{

Console.WriteLine("Друк крапки ({0}, {1})", this.X, this.Y);

}

void IPrint.PrintPreview()

{

Console.WriteLine(

"Перегляд перед друком крапки ({0}, {1})", this.X, this.Y);

}

void IMail.SendMail(string mailAddress)

{

Console.WriteLine(

"Відправка крапки ({0}, {1}) за адресою {2}",

this.X, this.Y, mailAddress); }}

Той факт, що клас реалізує ті або інші інтерфейси, відбивається в ключовому слові *class*. Назви інтерфейсів, що реалізуються, перераховуються після цього слова через кому:

class Point : IPrint, IMail

Якби наш клас Point був успадкований від базового класу з ім'ям, наприклад, *Shape* і додатково реалізовував інтерфейси *IPrint і IMail*, це можна було б записати таким чином:

class Point : Shape, IPrint, IMail

Проте простого перерахування інтерфейсів, що реалізовуються, недостатньо. У класі необхідно розташувати тіло методів цих інтерфейсів. У нашому випадку клас *Point* повинен містити тіло всіх методів інтерфейсів *IPrint і IMail*.

Зверніть увагу, як ми визначили метод *Print* інтерфейсу *IPrint*:

void IPrint.Print ()

Console.WriteLine("Друк крапки ({0}, {1})", this.X, this.Y);

Ім'я методу вказане як *IPrint. Print*. Тут ми забезпечили ім'я методу префіксом у вигляді імені інтерфейсу. Хоча такий префікс необов'язковий, все ж краще його вказувати. Це дозволить уникнути неоднозначності, якщо клас реалізує інтерфейси з однаковими назвами методів. Наприклад, якби в інтерфейсах *IPrint і IMail* був визначений метод, при реалізації методів ми могли б їх розрізняти по повних іменах *IPrint.Check* і *IMaiI.Check* відповідно.

## **Комбіновані інтерфейси**

Як ми вже говорили, неможливість множинного наслідування класів в С# з успіхом компенсується наявністю потужного механізму інтерфейсів. Похідний клас може бути успадкований лише від одного базового класу, проте при цьому він може реалізувати довільну кількість інтерфейсів.

Інтерфейси групують описи наборів методів, що мають схоже призначення і функціональність. При необхідності можна комбінувати інтерфейси, створюючи нові інтерфейси на базі уже існуючих.

Визначаємо інтерфейси *IChannel*, *IVolume* і *ITunning*, призначені для управління каналами і гучністю звуку:

interface IChannel

{

void switchTo(uint channelNumber);

uint current();

}

interface IVolume

{

void setLevel(uint volumeLevel);

uint current();

}

interface ITunning

{

void next();

void prev();

uint current();

}

Деякі з цих інтерфейсів (*IChannel* і *IVolume*) застосовуються для керування телевізором, а деякі (*ITunning* і *IVolume*) — для управління радіоприймачем.

З метою спрощення програми ми попарно скомбінували інтерфейси, створивши нові інтерфейси *ITvControl* і *IRadioControl*:

interface ITvControl : IChannel, IVolume

{

void on();

void off();

interface IRadioControl : ITunning, IVolume

Перший з цих інтерфейсів з ім'ям *ITVControl* об'єднує інтерфейси *IChannel* і *IVolume*, додаючи до них ще два методи (*on* і *off*), що дозволяють включати і вимикати телевізор. Таким чином, цей комбінований інтерфейс призначений для управління телевізором.

Комбінований інтерфейс *IRadioControl* є чистою комбінацию інтерфейсів *ITunning* і *IVolume* без яких-яких доповнень. Він створений для управління радіоприймачем.

Тепер при оголошенні класу *TvSet* нам досить вказати, що він реалізує комбінований інтерфейс *ITvControl*, не перераховуючи складові його інтерфейсу *IChannel* і *IVolume*:

class TvSet : ITvControl

{

private uint channel;

private uint volume;

private bool power;

public TvSet()

{

channel = 1;

volume =10;

power = false;

void ITvControl .on ()

{

power = true;

}

void ITvControl.off()

{

power = false;

}

…

}

Оскільки комбінований інтерфейс *ITvControl* включає методи *on* і *of f*, нам необхідно реалізувати їх в класі *TvSet*. Для зберігання поточного стану телевізора (включений або вимкнений) ми передбачили в цьому класі поле *power*.

Аналогічно при оголошенні класу *RadioSet* ми вказали, що він реалізує комбінований інтерфейс *IRadioControl*:

class RadioSet : IRadioControl

{

private uint channel;

private uint volume;

public RadioSet(;

{

channel = 1;

volume = 10;

}

…

}

Оскільки в рамках цього інтерфейсу немає жодних додаткових методів, відсутніх в складових його інтерфейсах, при оголошенні класу *RadioSet* нам не довелося оголошувати додаткові методи.

Тепер поговоримо про те, як користуватися комбінованими інтерфейсами.

Створивши об'єкт класу *TvSet*, ми оголошуємо змінну *tvControl*, призначену для зберігання посилання на інтерфейс *ITvControl*:

TvSet tv;

tv = new TvSet();

ITvControl tvControl = tv as ITvControl;

Ця змінна ініціалізувалася безпечним способом із застосуванням оператора *as*. Якщо посилання на інтерфейс *ITvControl* отримане успішно, ми використовуємо його для виклику методів комбінованого інтерфейсу:

if(tvControl != null)

{

tvControl.on();

tvControl.switchTo(10);

tvControl.setLevel(50) ;

tvControl.off();

}

else

Console.WriteLine("He реалізований інтерфейс ITvControl");

Зверніть увагу, що за допомогою єдиного посилання на комбінований інтерфейс *ITvControl* ми викликаємо методи вхідних в нього інтерфейсів *IChannel* і *IVolume*. Оскільки всі ці методи називаються по-різному, в даному випадку не виникають конфлікти між однаковими іменами методів, що відносяться до різних інтерфейсів.

Аналогічним чином створюється і використовується інтерфейс *IRadioControl* призначений для управління радіоприймачем:

RadioSet radio;

radio = new RadioSet();

IRadioControl radioControl = radio as IRadioControl;

if (radioControl != null)

{

radioControl.next();

radioControl.setLevel(30) ;

}

else

Console.WriteLine("He реалізований інтерфейс IRadioControl");

А тепер ми розглянемо цікавіший випадок, зв'язаний з викликом методу *current*. Цей метод, що повертає поточне значення номера каналу або рівня гучності, визначений у всіх базових інтерфейсах, наявних в нашій програмі.

Спроби звернення до цього методу через посилання на інтерфейси вигляду *tvControl.current ()* і *radioControl. current ()* приведуть до появи на етапі компіляції повідомлення про помилку. Річ у тому, що в такому записі не ясно, про яку саме реалізацію методу current йде мова.

Насправді, комбінований інтерфейс *ITvControl* складається з інтерфейсів *IChannel* і *IVolume*. У кожному з них є свій метод *current*. Аналогічно комбінований інтерфейс *IRadioControl*, що складається з інтерфейсів *Itunning і IVolume*, теж містить в собі дві різні реалізації методу current.

Для того, щоб уникнути неоднозначності, необхідно наводити посилання на комбінований інтерфейс до типу того інтерфейсу, для якого потрібно викликати наш метод:

Console.WriteLine("Телевізор: канал {0}, гучність {1}",

((IChannel)tvControl).current(),

((IVolume)tvControl).current());

Тут ми спочатку викликаємо метод *current* інтерфейсу *IChannel* (отримуючи при цьому поточний номер каналу), а потім цей же метод, але для інтерфейсу *IVolume* (для визначення поточного рівня гучності).

Аналогічним чином ця операція виконується і для інтерфейсу *IradioControl*, що керує радіоприймачем:

Console.WriteLine("Радіо: канал {0}, гучність {1}",

((ITunning)radioControl).current!),

((IVolume)radioControl).current());

Для того, щоб уникнути виникнення виключень, для перетворення типів ви можете скористатися описаними вище операторами *is* і *as*.

## **Інтерфейси і наслідування класів**

Якщо базовий клас реалізує будь-які інтерфейси, то вони успадковуються похідними класами. При необхідності похідний клас може перевизначити всі або деякі методи інтерфейсів базового класу.

Розглянемо простий приклад.

Нехай у нас є базовий клас *TvSet*, що описує телевізор. У цьому класі ми передбачили засоби перемикання каналів і регулювання гучності, що реалізуються за допомогою інтерфейсів.

Тепер на базі класу *TvSet* нам би хотілося створити новий клас *InternationalTvSet*, в якому парні канали були б російськими, а непарні— англійськими.

Для перемикання каналів і регулювання гучності ми визначили вже знайомі вам інтерфейси *IChannel* і *IVolume*:

interface IChannel

{

void switchTo(uint channelNumber);

uint current () ;

interface IVolume

void setLevel(uint volumeLevel);

uint current();

}

Ці інтерфейси реалізовані в базовому класі *TvSet* аналогічно тому, як це було зроблено в попередній програмі. Що ж до похідного класу *InternationalTvSet*, то він реалізує ще один інтерфейс *ILanguage*, що дозволяє визначити національну мову поточного каналу:

Interface ILanguage

{

string current();

}

При оголошенні похідного класу *InternationalTvSet* ми вказали базовий клас *TvSet*, а також перерахували всі інтерфейси, що реалізовуються в рамках базового і похідного класу:

class InternationalTvSet : TvSet, IChannel, IVolume, ILanguage

{

private string interfaceLanguage;

public InternationalTvSet()

{

interfaceLanguage = "російський";

}

void IChannel.switchTo(uint channelNumber)

{

channel = channelNumber;

if (channelNumber % 2 == 0)

interfaceLanguage = "російський";

else

interfaceLanguage = "english";

}

string ILanguage.current()

{

return interfaceLanguage;

}

}

Конструктор класу *InternationalTvSet* встановлює за умовчанням російську мову, ініціалізувавши відповідним чином поле *interfaceLanguage*. Надалі вміст цього поля залежатиме від номера каналу, що приймається телевізором.

У інтерфейсі *IChannel*, реалізованому базовим класом, немає засобів зміни вмісту поля *interfaceLanguage*, тому нам потрібно в похідному класі перевизначити метод *IChannel. switchTo*.

Нова реалізація цього методу, передбачена в похідному класі *InternationalTvSet*, приховує реалізацію цього ж методу базовим класом.

Додатково клас *InternationalTvSet* реалізує метод *current* інтерфейсу *ILanguage*, призначений для визначення поточної мови.

Тепер поговоримо про те, як метод *Main* нашої програми використовує описані вище інтерфейси.

Перш за все цей метод створює об'єкт похідного класу *InternationalTvSet* і отримує посилання на всі необхідні інтерфейси:

InternationalTvSet tv;

tv = new InternationalTvSet();

IChannel tvChannel = tv as IChannel;

IVolume tvVolume = tv as IVolume;

ILanguage tvLanguage = tv as ILanguage;

Якщо всі ці посилання отримані успішно, програма продовжує роботу, а в інакшому випадку виводить на консоль повідомлення про помилку:

if(tvChannel != null && tvVolume != null && tvLanguage != null)

{

…

}

else

Console.WriteLine(

"He реалізований інтерфейс IChannel, IVolume або ILanguage");

В тому випадку, якщо посилання на всі інтерфейси отримані успішно, програма встановлює рівень гучності 50 %, перемикає телевізор на 10-й канал, а потім відображає на консолі його стан:

tvVolume.setLevel(50);

tvChannel.switchTo(10);

Console.WriteLine("Телевізор: канал {0} ({1}), гучність {2}",

tvChannel.current(), tvLanguage.current(), tvVolume.current());

Оскільки номер 10-го каналу парний, для нього буде вибраний російський інтерфейс.

На наступному етапі програма перемикає телевізор на 11-й (непарний) канал повторюючи виведення стану на консоль:

tvChannel.switchToA1);

Console.WriteLine("Телевізор: канал {0} ({1}), гучність {2}",

tvChannel.current(), tvLanguage.current(), tvVolume.current());

Зверніть увагу, що тут ми використовуємо без будь-яких змін інтерфейс *IVolume*, реалізований в базовому класі *TvSet*.

Звертаючись до інтерфейсу *IChannel*, наша програма працює з методами цього інтерфейсу, реалізованими як в базовому, так і в похідному класі.

Таким чином, похідні класи можуть успадковувати не лише методи, але і інтерфейси базового класу, при необхідності перевизначаючи їх.

## **Властивості в інтерфейсах**

У попередніх прикладах програм ми створювали інтерфейси, що містять тільки методи. Тим часом в рамках інтерфейсів С# допускається оголошувати властивості, індексатори і події.

У цьому розділі ми розглянемо приклад програми управління телевізором, функцінування якої засновано на використанні властивостей, оголошених в рамках інтерфейсу *ITvControl*

interface ITvControl

{

bool PowerOn

{

get; set;

}

byte MaxChannel

{

get;

}

byte Channel

{

get; set;

}

byte Volume

{

get; set;

}

}

Властивість *PowerOn* дозволяє включати або вимикати телевізор за допомогою запису в нього значень *true* або *false* відповідно. Ця ж властивість може бути використано для визначення стану, в якому знаходиться телевізор, — включений чи вимкнений.

За допомогою властивості *MaxChannel* можна визначити максимальний номер каналу, що приймається телевізором. Ця властивість доступна лише для читання, оскільки у ній передбачена лише одна функція доступу *get*.

Властивість *Channel* дозволяє перемикати телевізор на будь-який заданий канал а також визначати номер поточного каналу.

І нарешті, за допомогою властивості *Volume* можна регулювати рівень гучності а також визначати поточний рівень гучності.

При оголошенні класу *TelevisionSet* ми вказали, що він реалізує тільки що описаний інтерфейс *ITVControl*:

class TelevisionSet : ITvControl

{

private bool isPowerOn;

private byte maxChannel;

private byte currentChannel;

private byte currentVolume;

public TelevisionSet(byte numberOfChannels)

{

isPowerOn = false;

maxChannel = numberOfChannels;

currentChannel = 1;

currentVolume = 10;

}

}

Зверніть увагу, що при реалізації інтерфейсу ми не вказуємо модифікатор доступу, а ім'я властивості задаємо разом з ім'ям інтерфейсу:

byte ITvControl.Channel

{

get

{

return currentChannel;

}

set

{

if(value <= maxChannel && value > 0)

currentChannel = value;

}

}

## **Індексатори в інтерфейсах**

Якщо клас створений для представлення об'єкту з масивом, для доступу до нього можна використовувати індексатори. Індексатор може бути включений в оголошення інтерфейсу поряд з методами і іншими членами.

Для отримання доступу до елементів масиву назв каналів ми оголошуємо інтерфейс *IChannelNames*:

interface IChannelNames

{

uint Size

{

get;

}

string this[uint index]

{

get; set;

}

}

Цей інтерфейс включає властивість *Size*, а також індексатор.

Властивість *Size* дозволяє взнати поточний розмір масиву, створеного конструктором, тому воно має лише одну процедуру доступу *get*. Що ж до індексатора, то ми оголосили для нього обидві процедури доступу — *get* і *set*.

Таким чином, через інтерфейс *IChannelNames* програма зможе визначити розмір масиву, а також отримати або змінити вміст його окремих комірок.

При оголошенні класу *ChannelNames* ми вказали, що він реалізує інтерфейс *IChannelNames*:

class ChannelNames : IChannelNames

{

private string[] Channels;

private uint ArraySize;

public ChannelNames(uint Count)

{

Channels = new string[Count];

ArraySize = Count;

}

…

}

Конструктор цього класу забезпечує створення масиву необхідних розмірів а також ініціалізацію поля *ArraySize*, що зберігає розмір масиву.

Реалізація властивості *Size* не має жодних особливостей:

uint IChannelNames.Size

{

get

{

return ArraySize;

}

}

Тут використовується повне ім'я властивості, що включає ім'я інтерфейсу; модифікатора доступу не вказується.

Що ж до індексатора, то він реалізований таким чином:

string IChannelNames.this[uint index]

{

get

{

if(index >= 0 && index < Channels.Length)

return Channels[index];

else

return "Канал недоступний";

}

set

{

if(index >= 0 && index < Channels.Length)

Channels[index]= value;

}

}

Ми додали до імені індексатора назву інтерфейсу, внаслідок чого получичи повне ім'я *IChannelNames.this*.

Тепер розповімо про те, як користуватися створеним нами інтерфейсом.

Спочатку необхідно створити об'єкт (масив назв каналів), а також посилання на інтерфейс:

ChannelNames ch = new ChannelNamesE);

IChannelNames chNames = ch as IChannelNames;

Створене посилання потім перевіряється на рівність значенню null:

if(chNames == null)

Console.WriteLine("He реалізований інтерфейс IChannelNames");

Якщо посилання на інтерфейс було отримане успішно, ми ініціалізували 5 комірок масиву, використовуючи посилання на інтерфейс

IChannelNames:

chNames[0]= "Спорт";

chNames[1]= "Світ кіно";

chNames[2]= "Бойовик";

chNames[3]= "Кіно";

chNames[4]= "MTV";

Як бачите, адресація комірок масиву через індексатор, оголошений в рамках інтерфейсу, виконується очевидним і очікуваним чином.

Читання комірок масиву відбувається аналогічно:

for(uint i = 0; i < chNames.Size; i++)

{

string s = chNaraes[i];

Console.WriteLine (s);

}

Перебираючи комірки, ми отримуємо черговий рядок назви каналу і виводимо його на консоль.

# **Хід роботи**

**Перша частина**

* Реалізувати ланцюжок наслідування у якому б був звичайний клас, абстрактний клас та інтерфейс. Перелічіть відмінності та подібності у цих структурах у звіті у вигляді таблиці.
* Реалізувати різні модифікатори доступу. Продемонструвати доступ до цих модифікаторів там де він є, та їх відсутність там, де це заборонено (включити в звіт вирізки з скріншотів Intelisense з VisualStudio).
* Реалізувати поля та класи без модифікаторів доступу. Дослідити який буде доступ за замовчуванням у класів, структур інтерфейсів, вкладених класів, полів, і т.д. У звіті має бути відповідна таблиця.
* Оголосити внутрішній клас з доступом меншим за public. Реалізувати поле цього типу даних. Дослідити обмеження на модифікатор.
* Реалізувати перелічуваний тип. Продемонструвати різні булівські операції на перелічуваних типах(^,||, &&. &, |,…).
* Реалізувати множинне наслідування у C#.
* Реалізувати перевантаження конструкторів базового класу та поточного класу. Показати різні варіанти використання ключових слів base та this. Реалізувати перевантаження функції.
* Реалізувати різні види ініціалізації полів як статичних так і динамічних: за допомогою статичного та динамічного конструктора, та ін. Дослідити у якій послідовності ініціалізуються поля.
* Реалізувати функції з параметрами out, ref. Показати відмінності при наявності та без цих параметрів. Показати випадки, коли ці параметри не мають значення.
* Продемонуструвати boxing / unboxing
* Реалізувати явні та неявні оператори приведення то іншого типу (implicit та explicit)
* Реалізувати логіку свого завдання у системі класів.
* Скопіювати проект і перейменувати всі класи у структури. Дослідити відмінності у класах та структурах та записати їх у вигляді таблиці до звіту. Реалізувати наслідування структур через інтерфейси
* Перевизначити і дослідити методи класу object (у тому числі і protected методи)

**Друга частина**

Дослідити швидкодію структур у відповідності з завданням.

Розглянемо, для прикладу, завдання у порівнянні структур і класів. Для порівняльного аналізу створюємо клас і структуру з ідентичними полями. У програмі створюємо велику кількість об’єктів одного та іншого типу та вимірюємо зміни оперативної пам’яті, часу ініціалізації, тощо.

За замовчуванням для експериментів брати 10 млн однакових операцій, але якщо виникають проблеми з швидкодією чи визначенням часу, то орієнтуватись до можливостей власного комп’ютера .

Потрібно провести кілька обчислень у різний час (оскільки інші процеси можуть вплинути на результат) і узяти середнє значення відповідних вихідних даних.

Для вимірювання часу рекомендується використати клас **Stopwatch**.

**У звіті повинна міститись наступна інформація:**

1) Дані комп'ютера з параметрами на яких відбувалось тестування;

2) Дані 5 обчислювальних експериментів (час потрібний для виконання операцій та оперативна пам’ять);

3) Усереднені дані обчислювальних експериментів.

4) У висновках повинно бути обґрунтування отриманих результатів.

Приклад:

1) Дані комп'ютера з параметрами на яких відбувалось порівняльне тестування швидкодії та потреб оперативної пам'яті для структур та класів:

Процесор – Інтел 8 Hz, ОЗП 2 Гб.

2) Дані чисельних експериментів

|  | Час створення  10 млн. структур, c | Час створення  10 млн. класів, c | Час створення  10 млн. наслідуваних класів, c | …. |
| --- | --- | --- | --- | --- |
| Експеримент №1 | 2.56 | 3.56 | 2.56 |  |
| Експеримент №2 | 2.53 | 3.51 | 2.53 |  |
| Експеримент №3 | 2.53 | 3.56 | 2.53 |  |
| …. | … | … | … |  |
| Середнє значення | … | … | … |  |

Додатково додати скріншоти диспетчера задач, якщо створювались великі масиви даних.

3) Усереднені дані обчислювальних експериментів

|  | Для структур | Для класів |
| --- | --- | --- |
| Середній час викликів методів, 10 млн. об'єктів, MB | 100 | 230 |
| Середній час створення  10 млн. структур, с | 2.53 | 1.53 |
| Середній час 10 млн. арифметичних операцій у об'єктах, с | 1.22 | 1.25 |
| … | … | … |

4) Висновки. Із результатів випливає, що операції з структурами приблизно однакові по швидкості, тому що… Час ініціалізації класів більший, тому що …. Час знищення класів більший, тому що ….

# **Варіанти завдань**

1. Реалізувати поліморфізм через інтерфейси, рефлексію та звичайним способом. Порівняти швидкість виконання поліморфізму за допомогою віртуальних методів і за допомогою інших способів. Як відомо, при виклику віртуального методу, викликається метод відповідно до типу об’єкта. Наприклад, якщо є базовий клас BaseClass і клас нащадок InheritedClass які мають віртуальний метод *DoSmth*(), і відбулось присвоєння *BaseClass test = new InheritedClass(),* то при виклику *test.DoSmth*() викличеться метод класу InheritedClass , а не BaseClass. Цей самий результат можна досягнути за допомогою рефлексії або інтерфейсів. Реалізувати ланцюжок наслідування з 3 або 4-ох класів.

…

1. Викликайте ті самі методи але за допомогою рефлексії, порівняйте результати.
2. Як відомо, у мові програмування С# існує 3 можливі передачі параметрів у функцію, для прикладу void DoSmthg1(ref x), void DoSmthg2(out x), void DoSmthg3(x). Визначити швидкість виконання для усіх трьох способів передачі параметрів. У процедурі DoSmthg обчислити нове значення змінної x. Проведіть експеримент для змінної х повинна як вказівного типу (reference type) так і значимого типу (value type). Дослідити, як впливає кількість полів у типі даних “x” на результати експериментів.
3. Розглянути швидкість оперування методів та потреби стеку із різними модифікаторами доступу private, protected і.т.д. Дослідити, як впливає на швидкість те, що ці методи належать вкладеному (nested) класу.
4. Розглянути швидкість створення та знищення структур у порівнянні з класами у платформі C#. Також порівняти результати з закритими (sealed) класами. Використовувати ключове слово using для детерміністичного знищення даних.
5. Розглянути швидкість створення та знищення базових та наслідуваних класів у платформі C#. Дослідити ланцюжок наслідування принаймі з 4-ох класів. Використовувати ключове слово using для детерміністичного знищення даних.
6. Розглянути швидкість оперування та знищення класів у мові С# в порівнянні із об’єктами розробленими на мові C++ (розробленими не на платформі .Net). Використовувати ключове слово using для детерміністичного знищення даних у платформі .Net.
7. Розглянути швидкість оперування та знищення структур у мові С# в порівнянні із об’єктами розробленими на мові C++ (розробленими не на платформі .Net). Використовувати ключове слово using для детерміністичного знищення даних у платформі .Net.
8. Розглянути швидкість арифметичних операцій (\*, -, /) та потреби стеку на мові С# і C++ (розробленими не на платформі .Net). Розглянути швидкість операцій з символами/стрічками та потреби стеку на мові С# і C++ (розробленими не на платформі .Net)
9. Порівняти швидкість операцій boxing/unboxing та звичайного присвоєння для різних типів (int, double, float). Дослідити залежність часу операцій boxing/unboxing для структур із різною кількістю полів.
10. Розглянути швидкість виконання для різних ініціалізацій у класі: ініціалізації змінних у конструкторі, ініціалізації змінних поза конструктором, статичній ініціалізації змінних, ініціалізації змінних у статичному конструкторі.
11. Статичні класи. Порівняти час створення і потреби стеку для викликів методів статичного та звичайного класів. Порівняти час виконання для звичайних та статичних методів. Порівняти час доступу та операцій над звичайними та статичними змінними.
12. Порівняти час створення для викликів методу звичайним способом та через рефлексію. Порівняти час створення і потреби пам'яті для створення класів звичайним способом та через рефлексію.
13. Дослідити множинне наслідування інтерфейсів. Як впливає кількість наслідуваних інтерфейсів на час створення класу? Як впливає кількість полів у цих інтерфейсах на час створення класу?
14. Дослідити наслідування від абстрактних класів реалізувати ланцюжок наслідування abstract class A > abstract class B> concrete class C (можна доповнити ще кількома класами). Як впливає кількість перевизначених методів абстрактних класів на час створення класу? Як впливає кількість абстрактних базових класів на час створення класу.
15. Дослідити наслідування від абстрактних класів реалізувати ланцюжок наслідування abstract class A > abstract class B> abstract class C > concrete class D (можна доповнити ще кількома класами). Як впливає кількість абстрактних класів на час виконання віртуальних методів класу? Як впливає відстань у класах між оголошенням та реалізацією методів (порівняти методи оголошені у класах A, B та С, та реалізовані у D ). Реалізувати аналогічне наслідування через інтерфейси та порівняти отримані результати.

# **Контрольні запитання.**

1. Що таке множинне наслідування?
2. Які елементи є у інтерфейсу?
3. Які модифікатори доступу може мати інтерфейс
4. Чим відрізняється інтерфейс і абстрактний клас?
5. Чим подібні інтерфейс і абстрактний клас?
6. Чим відрізняється структура і клас у С#?
7. Чим подібні структура і клас у С#?
8. Що роблять оператори as та is?
9. Як точно виміряти час виконання програми?
10. Які є види реалізації інтерфейсу?
11. Як реалізувати інтерфейс?

**СПИСОК ЛІТЕРАТУРИ**

1. Трей Нэш. C# 2008. Ускоренный курс для профессионалов.- Издательский дом “Вильямс”, 2008.
2. А. В. Фролов, Г. В. Фролов. Язык C# -самоучитель - Издательство “Диалог-МИФИ”, Москва, 2003.
3. Эндрю Троелсен. C# и платформа .Net – Издательство “Питер”, 2004.
4. Герберт Шилдт. C# Учебный Курс. – Издательство “Питер”, 2003.
5. Paolo Pialors, Marco Russo. Introducing Micrisoft LINQ. – Microsoft Press, 2007.
6. Daniel Solis. Illustrated C# 2008.- Apress, 2008.

НАВЧАЛЬНЕ ВИДАННЯ

**ЗАСТОСУВАННЯ ЛОГІКИ ВИСЛОВЛЮВАНЬ В ПРОГРАМНІЙ ІНЖЕНЕРІЇ**

**МЕТОДИЧНІ ВКАЗІВКИ**

до виконання практичних завдань

з дисципліни „ Конструювання програмного забезпечення” ”

для студентів базового напряму

„ Програмна інженерія ”

*Укладачі*  Сердюк Павло Віталійович

Сівець Ольга

*Редактор*

*Комп’ютерне верстання*