

## Лабораторна робота № 4

### УСПАДКУВАННЯ В C#.

**Мета:** Набути умінь і навичок створення ієрархії класів C# у середовищі Microsoft Visual Studio 2022.

#### Короткі теоретичні відомості

У програмуванні успадкування дозволяє створювати новий клас на базі іншого. Клас, на базі якого створюється новий клас, називається базовим, а новий клас - спадкоємцем або похідним класом. В клас-спадкоємця з базового класу переходять поля, властивості, методи і інші члени класу.

Оголошення нового класу, який буде наслідувати інший клас, виглядає так:

```
class [ім'я_класу]: [ім'я_базового_класу] {  
    // Тіло класу  
}
```

**Приклад.** На основі базового класу Тварина створюються два класи – Собака і Кішка, у ці два класи переходить властивість *Ім'я тварини*:

```
class Animal  
{  
    public string Name { get; set;  
}  
}  
class Dog :  
    Animal  
{  
    public void  
    Guard()  
    {  
        Console.WriteLine("Собака охороняє");  
    }  
}  
class Cat :  
    Animal  
{  
    public void  
    CatchMouse()  
    {
```

```

        Console.WriteLine("Кішка ловить мишу");
    }
} class
Program
{
    static void Main(string[]
args)
    {
        Dog dog1 = new Dog();
        dog1.Name = "Барбос"; // Називаємо пса
        Cat cat1 = new Cat();
        cat1.Name = "Сірко"; // Називаємо кота
        dog1.Guard(); // Відправляємо пса охороняти
        cat1.CatchMouse(); // Відправляємо кота на полювання
    }
}

```

### *Виклик конструктора базового класу в C#*

У базовому класі і класі-спадкоємця можуть бути оголошені конструктори. Конструктор базового класу буде створювати ту частину об'єкта, яка належить базовому класу (адже з базового класу про спадкоємця нічого невідомо), а конструктор з спадкоємця буде створювати свою частину.

Коли конструктор визначений тільки в спадкоємця, то при створенні об'єкта спочатку викликається конструктор за замовчуванням базового класу, а потім конструктор спадкоємця.

Коли конструктори оголошені і в базовому класі, і в спадкоємця - необхідно викликати їх обидва. Для виклику конструктора базового класу використовується ключове слово **base**. Оголошення конструктора класу-спадкоємця з викликом базового конструктора має наступну структуру:

```

[ім'я_конструктора_класа-спадкоємця] ([аргументи]): base([аргументи])
{
    // Тіло конструктора }

```

У базовий конструктор передаються всі необхідні аргументи для створення базової частини об'єкта.

Наведемо приклад виклику базового конструктора. Є той же клас Тварина і клас Папуга. У класі Тварина є тільки властивість Ім'я, і конструктор, який дозволяє встановити це ім'я. У класі Папуга є властивість Довжина дзьоба і конструктор, в якому ми задаємо цю довжину. При створенні об'єкта Папуга ми вказуємо два аргументи - ім'я і дзьоб, і далі аргумент Ім'я передається в базовий конструктор, він викликається, і після його роботи виконання передається конструктору класу Parrot, де встановлюється довжина:

```
class Animal
{
    public string Name { get; set; }
}

public Animal(string name)
{
    Name = name;
}

class Parrot : Animal
{
    public double BeakLength { get; set; } // Довжина дзьобу
    public Parrot(string name, double beak) : base(name)
    {
        BeakLength = beak;
    }
}

class Dog :
Animal
{
    public Dog(string name) :
base(name)
    {
        // Тут може бути логіка створення об'єкту Dog
    }
}
```

```

    } class
    Program
    {
        static void Main(string[]
args)
        {
            Parrot parrot1 = new Parrot("Кеша", 4.2);
            Dog dog1 = new Dog("Барбос");
        }
    }

```

### *Доступ до членів базового класу з класу-спадкоємця*

Тут варто зазначити, що в класі-спадкоємці ми можемо отримати доступ до членів базового класу які оголошені як *public*, *protected*, *internal* і *protected internal*. Члени базового класу з модифікатором доступу *private* також переходять у клас-спадкоємець, але до них можуть мати доступ тільки члени базового класу. Наприклад, властивість, оголошена в базовому класі, яка управляє доступом до закритого поля, працюватиме коректно в класі спадкоємця, але окремо отримати доступ до цього поля з класу спадкоємця ми не зможемо.

### *Поліморфізм в C#*

Віртуальний метод - це метод, який може бути перевизначений в класі спадкоємця.

Перевизначення методу - це зміна його реалізації в класі спадкоємця. Перевизначивши метод, він працюватиме по-різному в базовому класі і класі спадкоємця, маючи при цьому одне і те ж ім'я та аргументи і тип повернення.

*Віртуальний метод* оголошується за допомогою ключового слова ***virtual***:

```

[модифікатор доступу] virtual
[тип][ім'я методу] ([аргументи]) {
    // Тіло методу
}

```

\* Статичний метод не може бути віртуальним.

Оголосивши віртуальний метод, ми тепер можемо перевизначити його в класі спадкоємця. Для цього використовується ключове слово `override`:

```
[модифікатор доступу] override  
[тип][ім'я методу] ([аргументи]) {  
    // Нове тіло методу  
}
```

Приклад. Є клас Людина, і від нього успадковуються ще два - Студент і Учень. У базовому класі є віртуальний метод *ShowInfo*, який виводить інформацію про об'єкт. У класах Студент і Учень цей метод перевизначається для того, щоб до висновку базової інформації додати ще специфічну, що відноситься до відповідного класу:

```
class Person  
{  
    public string Name { get; set;  
    }  
    public int Age { get; set; }  
  
    public Person(string name, int age)  
    {  
        Name = name;  
        Age = age;  
    }  
  
    public virtual void ShowInfo() // оголошення віртуального методу  
    {  
        Console.WriteLine("Людина \nІмя:" + Name + "\n" + "Вік:" + Age + "\n");  
    }  
}  
  
class Student :  
    Person  
{  
    public string HighSchoolName { get; set;  
    }  
  
    public Student(string name, int age, string hsName) : base(name, age)  
    {
```

```

        HighSchoolName = hsName;
    }
    public override void ShowInfo() // перевизначення методу
    {
        Console.WriteLine("Студент \nІмя:" + Name + "\n" + "Вік:" + Age + "\n" + "Назва
ВНЗ:" + HighSchoolName + "\n");
    }
} class Pupil :
Person
{
    public string Form { get; set; }    public Pupil(string name,
int age, string Form) :base(name, age)
    {
        this.Form = Form;
    }
    public override void ShowInfo() // перевизначення методу
    {
        Console.WriteLine("Студент (ка) \nІмя:" + Name + "\n" + "Вік:" + Age + "\n" +
"Клас:" + Form + "\n");
    }
} class
Program
{
    static void Main(string[]
args)
    {
        List<Person> persons = new List<Person>();
        persons.Add(new Person("Василь", 32));
        persons.Add(new Student("Андрій", 21, "МДУ"));
        persons.Add(new Pupil("Олена", 12, "7-Б"));
        foreach (Person p in persons)      p.ShowInfo();
        Console.ReadKey();
    }
}

```

У методі `main` ми створюємо список людей, в який додаємо просто людей, студента і учня, і далі виводимо інформацію про кожного з них викликом методу `ShowInfo()`. Результат роботи - вивід інформації відповідно типу об'єкта.

Що буде, якщо прибрати перевизначення, відкинувши ключові слова `virtual` і `override`? У такому випадку, в базовому класі і в класі спадкоємця будуть методи з однаковим ім'ям `ShowInfo()`. Програма працювати буде, але про кожен об'єкт, незалежно це просто людина або студент/учень, виводитиметься інформація тільки як про просту людину (буде викликатися метод `ShowInfo()` з базового класу).

Це можна виправити, додавши перевірки на тип об'єкту, і за допомогою приведення типів, викликати потрібний метод `ShowInfo()`:

```
foreach (Person p in persons)
{
    if (p is
Student)
        ((Student)p).ShowInfo();
    else if (p is Pupil)
        ((Pupil)p).ShowInfo();
    else
        p.ShowInfo();
}
```

Цей варіант коду є працездатним, але вимагає додаткові перевірки.

#### *Виклик базового методу*

Буває так, що функціонал методу, який перевизначається, в базовому класі мало відрізняється від функціоналу, який повинен бути визначений в класі спадкоємця. У такому випадку, при перевизначенні, ми можемо викликати спочатку цей метод з базового класу, а далі дописати необхідний функціонал.

Це робиться за допомогою ключового слова ***base***:

```
public virtual void ShowInfo() // ShowInfo в класі Person {
    Console.WriteLine("Ім'я:" + Name);
    Console.WriteLine("Вік:" + Age);
}

public override void ShowInfo() // ShowInfo в класі Student
```

```
{  
    base.ShowInfo(); // Викликає базовий метод ShowInfo ()  
    Console.WriteLine("Назва ВНЗ:" + HighSchoolName); }
```

### *Абстрактні класи*

Абстрактний клас - це клас оголошений з ключовим словом *abstract*:

```
abstract class [ім'я_класу] {  
    // тіло }
```

Такий клас має такі особливості:

- Не можна створювати екземпляри (об'єкти) абстрактного класу;
- Абстрактний клас може містити як абстрактні методи / властивості, так і звичайні;
- У класі спадкоємця повинні бути реалізовані всі абстрактні методи і властивості, оголошені в базовому класі.

Навіщо потрібні абстрактні класи?

У абстрактному класі, від якого ніхто не успадковується, сенсу немає, так як не можна створювати його екземпляри. В абстрактному класі зазвичай реалізується деяка загальна частина декількох сутностей або іншими словами - абстрактна сутність, яка, як об'єкт, не може існувати, і ця частина необхідна в класах спадкоємців. Конкретні приклади будуть далі.

### *Абстрактні методи*

Розуміння абстрактних методів базується на понятті віртуальних методів.

Абстрактний метод - це метод, який не має своєї реалізації в базовому класі, і він повинен бути реалізований в класі-спадкоємці. Абстрактний метод може бути оголошений тільки в абстрактному класі.

Різниця між віртуальним і абстрактним методом полягає в наступному:

- Віртуальний метод може мати свою реалізацію в базовому класі, абстрактний - ні (тіло порожнє);



- Абстрактний метод повинен бути реалізований в класі - спадкоємці, віртуальний метод перевизначати необов'язково.

Оголошення абстрактного методу відбувається за допомогою ключового слова *abstract*, і при цьому фігурні дужки опускаються, крапка з комою ставиться після заголовка методу:

```
[модифікатор доступу]  
abstract [тип][ім'я методу] ([аргументи]);
```

Реалізація абстрактного методу в класі спадкоємця відбувається так само, як і перевизначення методу - за допомогою ключового слова *override*:

```
[модифікатор доступу] override [тип][ім'я методу] ([аргументи]) {  
    // Реалізація методу  
}
```

### **Завдання до лабораторної роботи**

1. У консольному додатку побудувати ієрархію класів згідно варіанту.
2. Навести приклади перевантаження батьківських методів в класах нащадках. Для цього у базового класу повинно бути 2-3 методи для перевантаження.
3. Базовий клас зробити абстрактним.
4. Продемонструвати роботу методів *virtual* та *override*.
5. Перевірити роботу модифікаторів *new* та *sealed*.
6. Створити клас, якій містить список об'єктів утворених класів. Реалізувати в ньому метод для виклику *virtual* методів об'єктів, що належать одній ієрархічній структурі. Реалізувати методи для додавання та видалення об'єктів списку.
7. Створіть абстрактний базовий клас «Фігура» з абстрактними методами для підрахунку площі і малювання фігури. Створіть похідні класи: прямокутник, коло, прямокутний трикутник зі своїми реалізаціями методу для підрахунку площі та виводу фігури на екран. Для перевірки, визначте

масив посилань на абстрактний клас, за яким надаються адреси різних об'єктів класів нащадків та продемонструйте використання віртуальних методів.

### **Індивідуальні завдання**

1. Трансформатор, трансформатор струму, силовий трансформатор, автотрансформатор.
2. Студент, викладач, людина, доцент, декан.
3. Автомобіль, літак, корабель, транспортний засіб, легковий автомобіль.
4. Держава, республіка, монархія, королівство, демократія.
5. Службовець, персона, дитина, інженер, головний інженер.
6. Деталь, механізм, виріб, вузол, гвинт.
7. Організація, компанія, банк, страхова компанія, енергетична компанія.
8. Газета, журнал, книга, видання, підручник.
9. Тест, іспит, випробування, залік, диференційний залік.
10. Їжа, продукт, товар, хліб, послуга.
11. Квитанція, накладна, документ, рахунок, чек.
12. Автомобіль, поїзд, транспорт, експрес, автобус.
13. Електричний двигун, двигун, двигун внутрішнього згоряння, паровий двигун, асинхронний двигун.
14. Тварина, риба, ссавець, птах, примат.
15. Корабель, пароплав, вітрильник, фрегат, атомохід.
16. Держава, республіка, монархія, королівство, демократія.
17. Службовець, персона, дитина, інженер, головний інженер.
18. Деталь, механізм, виріб, вузол, гвинт.

19. Організація, компанія, банк, страхова компанія, енергетична компанія.

20. Газета, журнал, книга, видання, підручник.

Код програми (варіант 10):

```
using System;

namespace lab4_1
{
    sealed class Service(string shop_name)
    {
        private string _shop_name = shop_name;
        private List<Food> _food_list = new List<Food>();

        public void AddFood(Food food)
        {
            _food_list.Add(food);
        }

        public void RemoveFood(Food food)
        {
            _food_list.Remove(food);
            food.MoveToTrashBin();
        }

        public void RemoveAllFoods()
        {
            foreach (var food in _food_list)
            {
                food.MoveToTrashBin();
            }
            _food_list.Clear();
        }

        public void ShowAllFoods()
        {
            foreach (var food in _food_list)
            {
                Console.WriteLine(food.Name);
            }
        }
    }

    abstract class Product
    {
```

```

    public abstract string Name { get; set; }
    public abstract string Description { get; set; }
    public abstract double Price { get; set; }

    public abstract void Use();
    public abstract void Prepare();
}

class Food : Product
{
    public override string Name { get; set; }
    public override string Description { get; set; }
    public override double Price { get; set; }

    public override void Use()
    {
        Console.WriteLine($"Ви з'їли {Name}");
    }

    public override void Prepare()
    {
        Console.WriteLine($"Ви приготували {Name}");
    }

    public virtual void MoveToTrashBin()
    {
        Console.WriteLine($"Ви викинули {Name} у смітник");
    }
}

class Bread : Food
{
    public override void Prepare()
    {
        Console.WriteLine("Ви зробили грінку і приготували бутерброд");
    }

    public override void MoveToTrashBin()
    {
        Console.WriteLine("Ви не можете викинути хліб. Хліб – усьому голова");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Service service = new Service("Marketopt");
    }
}

```

```

        Food smetana = new Food();
        smetana.Name = "Yagotynske";
        Food food = new Food();
        food.Name = "Gingerbred";

        smetana.Use();

        Bread bread = new Bread();
        bread.Name = "kyivhlib";
        bread.MoveToTrashBin();

        service.AddFood(food);
        service.AddFood(smetana);

        service.ShowAllFoods();
        service.RemoveAllFoods();

    }
}

```

Результат роботи програми:

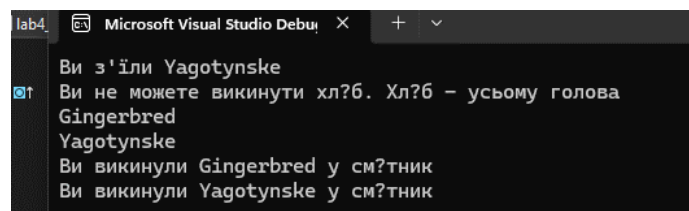


Рисунок 4.1 – Результат роботи програми з наслідуваними класами

Код до пункту 7(опис класів фігур):

```

using System;

namespace lab4_2
{
    class Program
    {
        public const double PI = 3.1415;
        public abstract class Figure
        {
            public abstract string name { get; set; }
            public abstract double GetArea();
        }
    }
}

```

```

        public abstract double GetPerimeter();
    }

    public class Square : Figure
    {
        public override string name { get; set; } = "Square";
        public double Height { get; set; }
        public double Width { get; set; }

        public override double GetArea()
        {
            return Height * Width;
        }

        public override double GetPerimeter()
        {
            return 2 * (Height + Width);
        }
    }

    public class Circle : Figure
    {
        public override string name { get; set; } =
"default_circle";

        public double Radius { get; set; }

        public override double GetPerimeter()
        {
            return Radius * 2 * PI;
        }

        public override double GetArea()
        {
            return PI * Radius * Radius;
        }
    }

    public class RightTriangle : Figure
    {
        public override string name { get; set; } = "default
triangle";

        public double Base { get; set; }
        public double Height { get; set; }
        public double Katet2 { get; set; }

        public override double GetArea()
        {

```

```

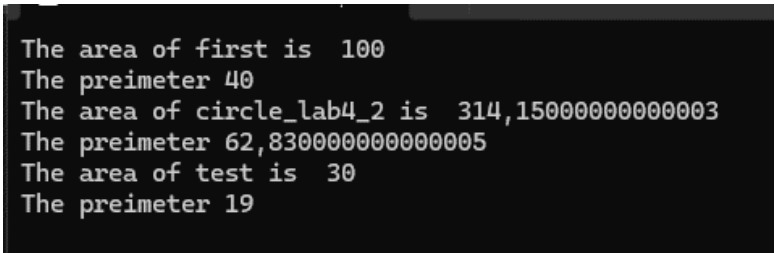
        return 0.5 * Base * Height;
    }

    public override double GetPerimeter()
    {
        return Base + Height + Katet2;
    }
}

static void Main(string[] args)
{
    Figure[] figures = new Figure[3];
    Square square = new Square();
    square.Height = 10;
    square.Width = 10;
    square.name = "first";
    figures[0] = square;
    Circle circle = new Circle();
    circle.name = "circle_lab4_2";
    circle.Radius = 10;
    figures[1] = circle;
    RightTriangle triangle = new RightTriangle();
    triangle.name = "test";
    triangle.Base = 10;
    triangle.Height = 6;
    triangle.Katet2 = 3;
    figures[2] = triangle;
    foreach (Figure f in figures)
    {
        Console.WriteLine($"The area of {f.name} is
{f.GetArea()}");
        Console.WriteLine("The preimeter " + f.GetPerimeter());
    }
}
}

```

Результат роботи програми:



```

The area of first is 100
The preimeter 40
The area of circle_lab4_2 is 314,15000000000003
The preimeter 62,830000000000005
The area of test is 30
The preimeter 19

```

Рисунок 4.2 – Результат роботи програми з класами геометричних фігур