ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Успадкування, поліморфізм

Лекція 10

План

1. Успадкування
2. Поліморфізм
3. Порівняння успадкування та поліморфізму
4. Приклади

Успадкування

Успадкування - один з принципів ООП, який дозволяє створювати новий клас на основі існуючого класу. Новий клас (нащадок, похідний клас) успадковує всі властивості та методи базового класу (батьківський клас, суперклас).

Базовий клас - визначає загальні характеристики та поведінку, які можуть бути успадковані похідними класами.

Клас-нащадок автоматично успадковує поля та методи базового класу.

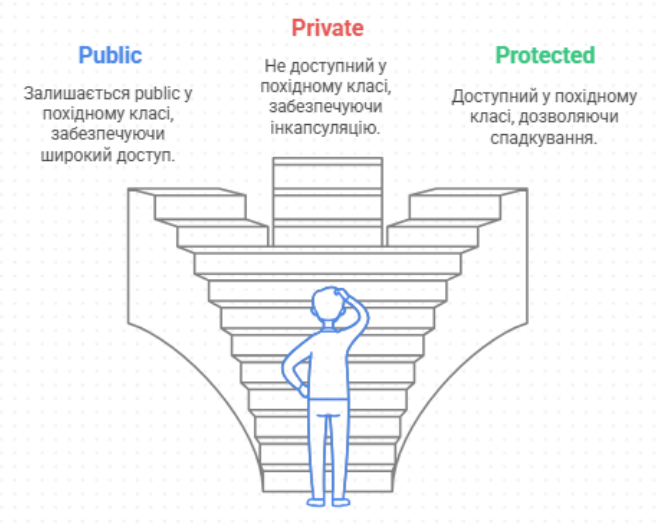
Успадкування

синтаксис успадкування:

```
[модифікатор] class <Ім’я похідного класу> : <ім’я базового класу>
{
        // Тіло похідного класу
}
```

Успадкування

Модифікатори доступу при успадкуванні



public члени *базового класу* залишаються **public** у похідному класі.

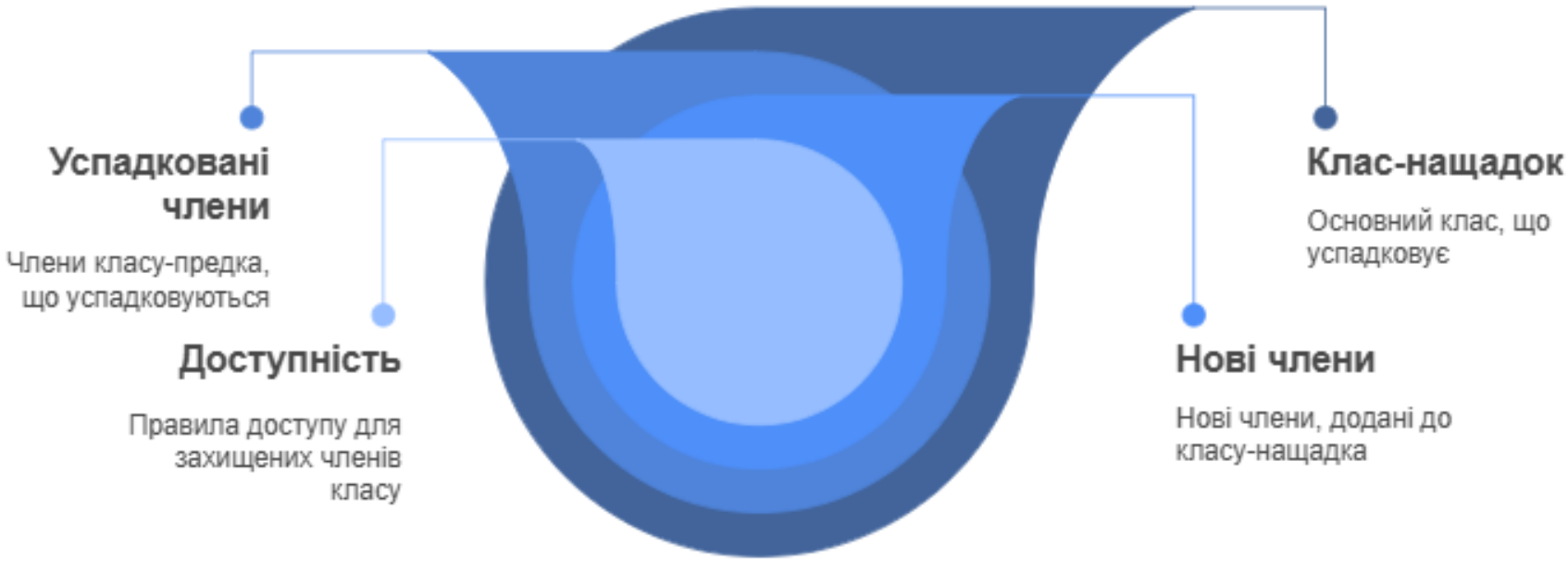
private члени *базового класу* не доступні в похідному класі.

protected члени *базового класу* доступні в похідному класі.

internal члени базового класу доступні в похідному класі, якщо вони знаходяться в тій самій збірці.

protected internal члени базового класу доступні в похідному класі, навіть якщо вони знаходяться в іншій збірці.

Успадкування



Клас-нащадок отримує у спадок усі члени (*окрім конструкторів*) класу-предка – це його, так звана, успадкована частина.

Клас-нащадок може містити і нові члени – це його власна частина.

У класі-нащадку доступ можливий тільки до тих членів успадкованої частини, які описані у класі-предку зі специфікаторами доступу *public* або *protected*.

Доступ до захищених (*protected*) членів успадкованої частини можливий тільки в межах опису класу-нащадка.

Успадкування

```
public class BaseClass
 {
     public int PublicA;
     private int PrivateB;
     protected int ProtectedC;
 }

 public class DerivedClass : BaseClass
 {
     public void AccessBaseClassMembers()
     {
         PublicA = 10;
         PrivateB = 20;     // Недоступно
         ProtectedC = 30;
     }
 }
```

Щоб зробити поле *PrivateB* доступним у класі *DerivedClass*, вам потрібно використовувати один із наступних підходів:

1. використати модифікатор *protected internal*;

2. використати модифікатор *internal*;

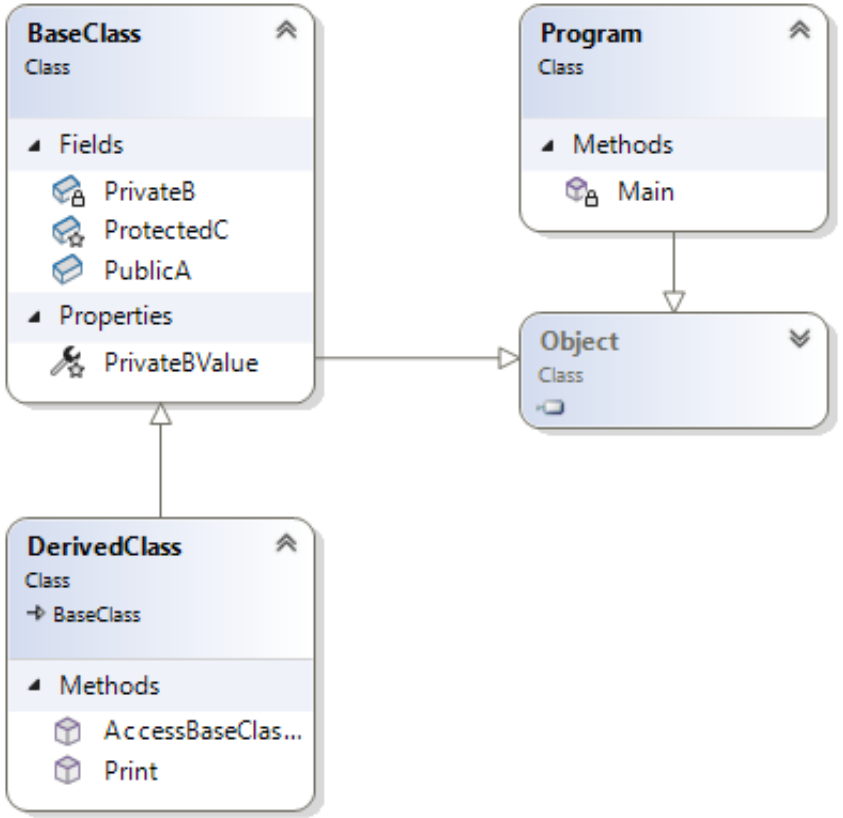
3. створити *protected* властивість.

Успадкування

```
public class BaseClass
    {
        public int PublicA;
        private int PrivateB;
        protected int ProtectedC;

        protected int PrivateBValue
        {
            get { return PrivateB; }
            set { PrivateB = value; }
        }
    }

public class DerivedClass : BaseClass
{
    public void AccessBaseClassMembers(){
        PublicA = 10;
        PrivateBValue = 20; // Доступ через властивість
        ProtectedC = 30;
    }
    public void Print(){
        Console.WriteLine($"PublicA: {PublicA}");
        Console.WriteLine($"PrivateB (через PrivateBValue): {PrivateBValue}");
         Console.WriteLine($"ProtectedC: {ProtectedC}");
        }
 }
```



Успадкування

```
DerivedClass derivedObject = new DerivedClass();

derivedObject.AccessBaseClassMembers();

derivedObject.Print();

BaseClass baseObject = new BaseClass();

baseObject.PublicA = 5;

Console.WriteLine($"BaseClass PublicA: {baseObject.PublicA}");
```

```
PublicA: 10
PrivateB (через PrivateBValue): 20
ProtectedC: 30
BaseClass PublicA: 5
```

Конструктори в похідних класах

За виділення пам’яті та ініціалізацію успадкованої частини класу-нащадка відповідає конструктор класу-предка, а за власну частину – конструктор класу-нащадка.
Тому будь-який конструктор класу-нащадка повинен забезпечити коректне конструювання як
базової частини, так і власної.

У мові C# є можливість явного виклику конструктора класу-предка

```
[модифікатор] <Ім’яКласу> ([<список форм. параметрів>]) : base ([<список факт. параметрів>])
{
                    // Тіло конструктору класу-нащадка
 }
```

Конструктори в похідних класах

При створенні об'єкта похідного класу, *спочатку викликається конструктор базового класу*, а потім - *конструктор похідного класу*. Це відбувається автоматично.

Якщо базовий клас має *конструктор за замовчуванням* (без параметрів), то він викликається автоматично.

Якщо базовий клас має *конструктор з параметрами*, то необхідно явно викликати його з конструктора похідного класу за допомогою ключового слова base.

Успадкування

```
BaseClass constructor: x = 10
DerivedClass constructor: y = 20
```

```
class BaseClass
{
    public BaseClass(int x)
    {
        Console.WriteLine($"BaseClass constructor: x = {x}");
    }
}

class DerivedClass : BaseClass
{
    public DerivedClass(int x, int y) : base(x)
    {
        Console.WriteLine($"DerivedClass constructor: y = {y}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        DerivedClass obj = new DerivedClass(10, 20);
    }
}
```

У прикладі при описі конструктора з параметрами у класі-нащадку ми змушені здійснити явний виклик конструктора з параметрами класу-предка, оскільки у класі предка відсутній конструктор без параметрів.

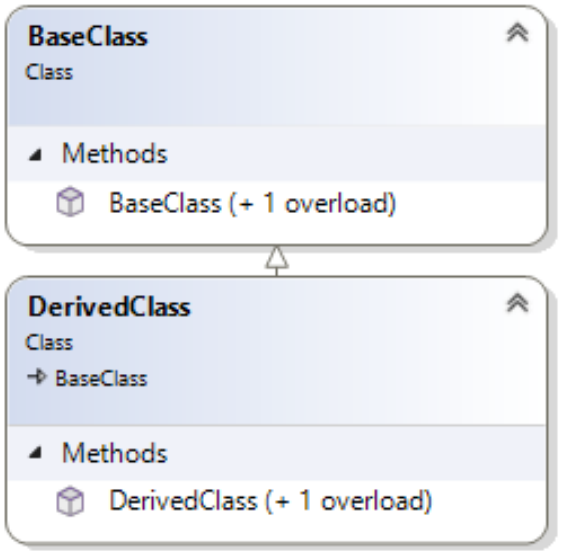
Приклад

```
class BaseClass{
      public BaseClass() {
          Console.WriteLine("Конструктор базового класу");
      }

      public BaseClass(int x){
          Console.WriteLine($"Конструктор базового класу з параметром
{x}");
      }
}

class DerivedClass : BaseClass{
      public DerivedClass() : base() {
        Console.WriteLine("Конструктор похідного класу");
      }

      public DerivedClass(int y) : base(y) {
        Console.WriteLine($"Конструктор похідного класу з параметром
{y}");
      }
}
```



Приклад

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

    public Animal(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public virtual void Sound()
    {
       Console.WriteLine("Animal makes a sound");
    }
}
```

Приклад

```
public class Dog : Animal
{
    public string Breed { get; set; }

    public Dog(string name, int age,
string breed) : base(name, age)
    {
        Breed = breed;
    }

    public override void Sound()
    {
        Console.WriteLine("Dog Woof");
    }
}
```

```
public class Cat : Animal
{
    public string Color { get; set; }

    public Cat(string name, int age,
string color) : base(name, age)
    {
        Color = color;
    }

    public override void Sound()
    {
       Console.WriteLine("Cat meows");
    }
}
```

Приклад

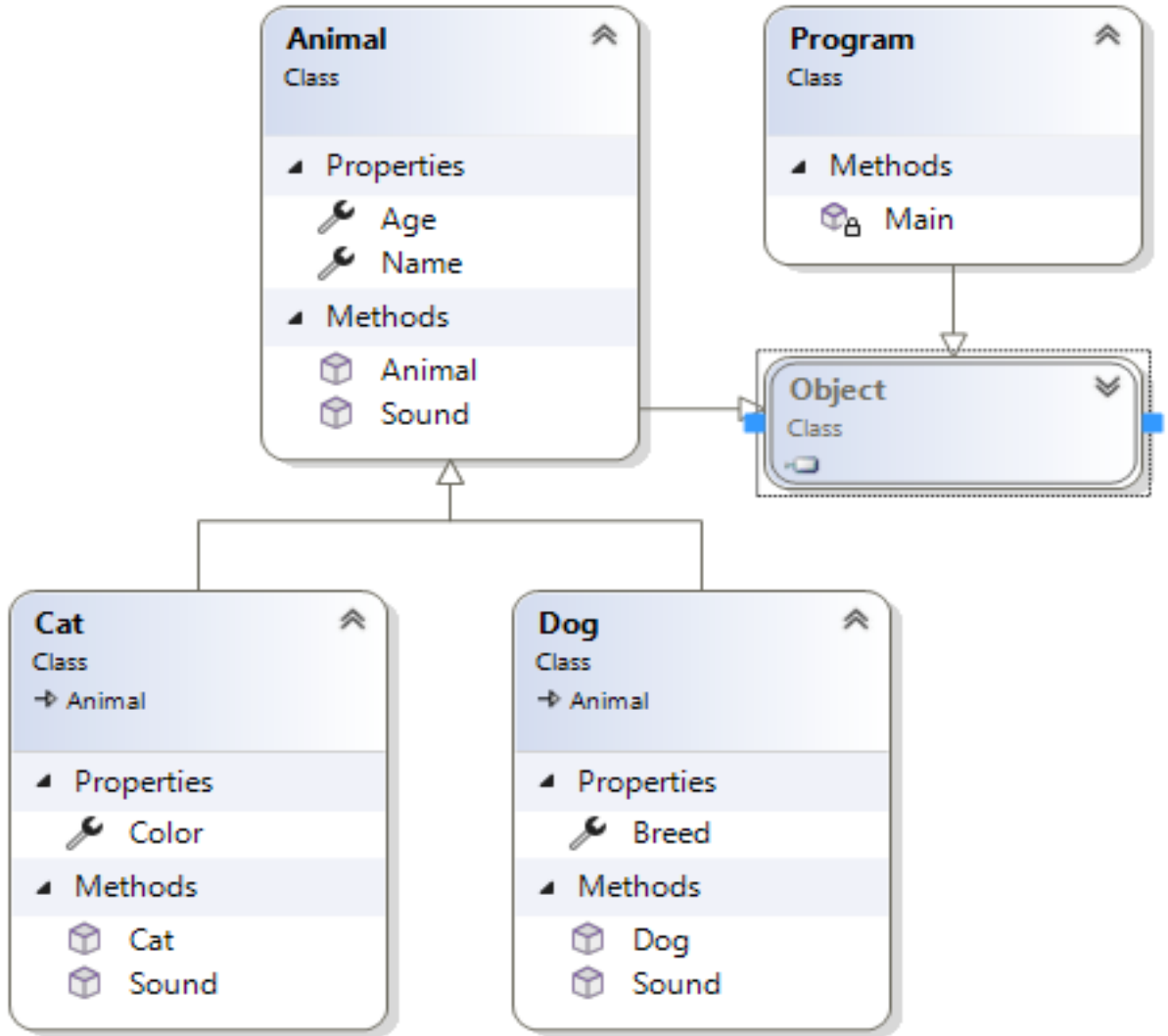
```
static void Main(string[] args)
{

    Dog dog = new Dog("Teddy", 3, "Golden Retriever");
    Cat cat = new Cat("Tess", 12, "Gray");

    Console.WriteLine($"{dog.Name} is a {dog.Breed} and is {dog.Age} years.");
    dog.Sound();

    Console.WriteLine($"{cat.Name} is a {cat.Color} cat and is {cat.Age} years.");
    cat.Sound();
}
```

Приклад


Animal
Class
Properties
Age
Name
Methods
Animal
Sound
Program
Class
Methods
Main
Object
Class
Cat
Class
Animal
Properties
Color
Methods
Cat
Sound
Dog
Class
Animal
Properties
Breed
Methods
Dog
Sound

Успадкування

Похідний клас успадковує всі відкриті (public) та захищені (protected) члени базового класу.

Приватні (private) члени базового класу не успадковуються.

Похідний клас може перевизначати методи базового класу за допомогою ключового слова new або override.

- new - приховує метод базового класу.
- override - надає нову реалізацію віртуального методу базового класу.

Конструктори базового класу не успадковуються.

Успадкування

Успадкування дозволяє використовувати вже написаний код батьківського класу, що зменшує обсяг коду та час розробки.

Можна додавати нові властивості та методи до похідного класу, не змінюючи батьківський клас.

Успадкування дозволяє створювати ієрархію класів, що відображає зв'язки між об'єктами.

Успадкування є основою для поліморфізму, який дозволяє об'єктам різних класів оброблятися однаково.

Успадкування

Заборона успадкування

Щоб заборонити успадкування класу в C#, використовується ключове слово sealed. Клас, оголошений як sealed, не може бути базовим класом для інших класів.

```
public sealed class MySealedClass
{
    public int MyProperty { get; set; }

    public void MyMethod(){
        Console.WriteLine("Метод класу
MySealedClass");
    }
}
// Спроба успадкування від MySealedClass призведе до
помилки компіляції
public class Program {

    public static void Main(string[] args){
        MySealedClass obj = new MySealedClass();
        obj.MyProperty = 10;
        obj.MyMethod();
    }
}
```

Поліморфізм

Поліморфізм у перекладі з грецької означає «наявність багатьох форм». Іншими словами
поліморфізм – це властивість деякого об'єкта приймати різні форми в залежності від ситуації. В ООП цей термін використовують по відношенню до методу. При цьому розрізняють три основні види поліморфізму:

1. Параметричний поліморфізм – дозволяє в межах одного класу описувати декілька методів з однаковим іменем але різним списком формальних параметрів. Відповідні методи називають поліморфними методами класу.
2. Простий поліморфізм – дозволяє перевизначити методи при успадкуванні. Відповідні методи називають статичними поліморфними.
3. Складний поліморфізм.

Поліморфізм

Параметричний поліморфізм (Перевантаження методів)

Параметричний поліморфізм дозволяє створювати методи з однаковим ім'ям, але різними типами або кількістю параметрів.

Компілятор вибирає відповідний метод на основі аргументів, які передаються при виклику.

Поліморфізм

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public double Add(double a, double b)
    {
        return a + b;
    }

    public string Add(string a, string b)
    {
        return a + b;
    }
}
```


```
Calculator calc = new Calculator();
int sumInt = calc.Add(5, 10);
Console.WriteLine(sumInt);

double sumDouble = calc.Add(3.5, 2.7);
Console.WriteLine(sumDouble);

string conk = calc.Add("Hello, ",
"World!");
Console.WriteLine(conk);
```

```
15
6,2
Hello, World!
```

Поліморфізм

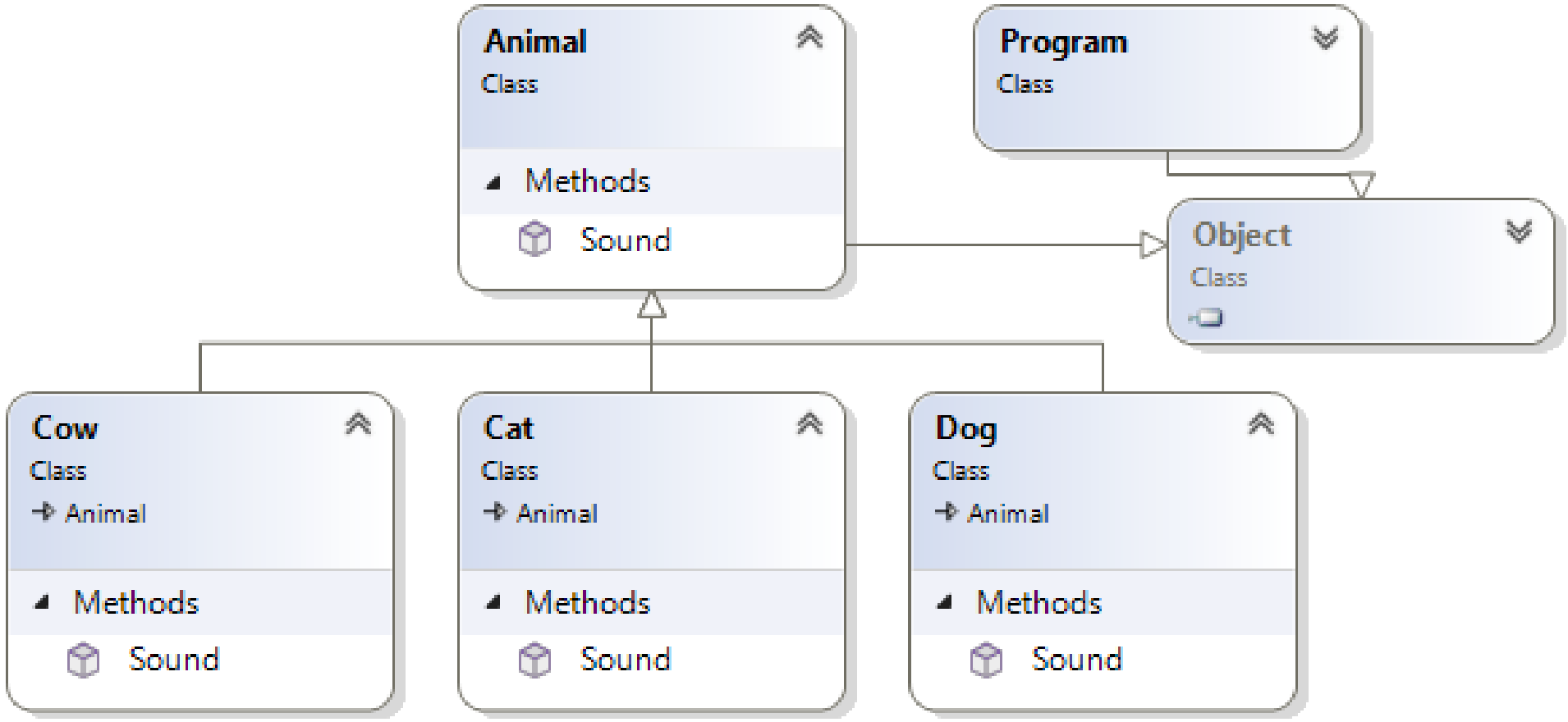
Простий поліморфізм

Поліморфізм дозволяє об'єктам різних класів, які мають спільний базовий клас, поводитися по-різному залежно від їхнього конкретного типу.

Поліморфізм часто реалізується через успадкування, коли похідні класи перевизначають методи базового класу за допомогою ключового слова override.
Методи, оголошені як virtual у базовому класі, можуть бути перевизначені в похідних класах.

Похідні класи можуть надавати власну реалізацію віртуальних методів базового класу, змінюючи їхню поведінку.

Поліморфізм



Поліморфізм

```
public class Animal {
    public virtual void Sound(){
        Console.WriteLine("Тварина видає звук");
    }
}
public class Cat : Animal {
    public override void Sound() {
        Console.WriteLine("Кішка нявкає");
    }
}
public class Dog : Animal {
    public override void Sound() {
        Console.WriteLine("Собака гавкає");
    }
}
public class Cow : Animal {
    public override void Sound() {
        Console.WriteLine("Корова мукає");
    }
}
```

```
Animal[] animals = new Animal[]
{
    new Cat(),
    new Dog(),
    new Cow()
};

foreach (Animal animal in animals)
    animal.Sound();
```

```
К?шка нявкає
Собака гавкає
Корова мукає
```

Поліморфізм

Складний поліморфізм

Складний поліморфізм (також відомий як узагальнення або дженерики) дозволяє створювати класи та методи, які можуть працювати з різними типами даних без необхідності писати окремий код для кожного типу.

```
public class GenericList<T>{
    private T[] items;
    private int count;
    public GenericList()    { items = new T[10]; count = 0; }

    public void Add(T item) {items[count] = item; count++;}

    public T Get(int index) {return items[index];}
}
…..
        GenericList<int> intList = new GenericList<int>();

        intList.Add(5);
        intList.Add(10);
        int firstInt = intList.Get(0);
        Console.WriteLine(firstInt);
        GenericList<string> stringList = new GenericList<string>();
        stringList.Add("Hello");
        stringList.Add("World");
        string firstString = stringList.Get(0);
        Console.WriteLine(firstString);
…..
```