

## 1. Спадкування

**Спадкування** – одне з трьох базових понять об'єктно-орієнтованого програмування (ООП). Інші два – **інкапсуляція** (приховування даних) і **поліморфізм** (один інтерфейс – багато реалізацій).

Спадкування передбачає багаторазове використання коду. Реалізовано на основі базових (батьківських) і похідних (дочірніх) типів.

### 1. Суть і умови успадкування

Кожен тип складається з членів типу: методів, властивостей, подій, полів і т.д., в яких реалізований функціонал типу. При спадкуванні типів успадковуються члени типу. Клас, від якого ведеться спадкування називається батьківським або базовим. Клас, який успадковує члени називається похідним або дочірнім.

У спадкуванні беруть участь тільки члени класи, інтерфейси, члени типів значень (структур) не успадковуються.

С # не підтримує множинне успадкування. Кожен клас може мати тільки один клас батько, але скільки завгодно дочірніх класів.

Спадкування в С # транзитивне. Якщо Class1 - базовий клас, а Class2 похідний від Class1, Class3 похідний від Class2, то Class3 успадковує всі члени Class1. Головний базовий тип, прабатько всіх інших - **System.Object**.

### 1.2 Механізм успадкування

По суті, похідний клас - це спеціалізація базового. Похідний клас реалізує власну функціональність на основі функціональності базового, тим самим змінюючи і розширюючи її.

Щоб вказати в коді, що один клас успадковується від іншого потрібно ключовий символ «:» між іменами базового і похідного класу.

```
class <ім'я_класу_нащадку> : <ім'я_базового_класу>
```

```
public class SomeClass : Object
    // Явне успадкування від типу Object
{
    // Визначаємо поля в базовому класі.
    public int firstclassfield;
    public string secondclassfield;
    // Визначаємо та реалізуємо конструктор.
    public SomeClass(int first, string second)
    {
        firstclassfield = first;
        secondclassfield = second;
    }
}
```

```

}

public class Class2 : SomeClass
    // Явне успадкування від SomeClass.
    //Клас спадкує усі поля та методи базового, крім конструктора
{
    public double thirdclassfield;
    // Розширяємо функціонал похідного класу. Додаєм нове поле
    // Визначаємо конструктор для похідного класу
    public Class2(int first,string second,double third)
    {
        thirdclassfield = third;
        firstclassfield = first;
        secondclassfield = second;
    }
}

```

### 1.3 Абстрактні базові класи і віртуальні методи

Якщо програміст оголосить базовий клас абстрактним, то всі члени в ньому (в тому числі і методи) також стануть абстрактними. В цьому випадку методи не потрібно визначати в батьківському класі, але необхідно - в дочірніх. Клас оголошується абстрактним за допомогою ключового слова **abstract** перед ім'ям класу в описі: **abstract class MyBestClass**. Абстрактні методи, успадковані від батьківського класу не потрібно визначати в дочірньому, якщо дочірній - теж абстрактний.

Якщо хоча б один член класу оголошений абстрактним, компілятор зробить весь клас абстрактним, навіть якщо це не зазначено програмістом.

Інший підхід до розширення функціональності пов'язаний з поліморфізмом і заснований на використанні віртуальних методів. Якщо метод базового класу програміст оголошує віртуальним, то він повинен визначити і реалізувати його. У похідному класі метод базового можна приховати за допомогою ключового слова **new** або перевизначити, використовуючи ключове слово **override** :

```

public class SomeMainClass
{
    // Оголошення членів
    public virtual void SomeMethod1()
    {
        // Код реалізації віртуального методу в базовому класі
    }
}

public class OtherClass : SomeMainClass
{
    // Оголошення членів типу дочірнього класу
    public override void SomeMethod1()

```

```

    {
        // Інша реалізація того ж методу в похідному класі
    }
    /* public new void SomeMethod1() - цей рядок у визначенні
        коду приховає віртуальний метод базового класу*/
}

```

## 1.4 Доступ до членів базового класу і заборона успадкування

Всі члени, крім статичних членів і конструкторів з деструкторами, успадковуються похідними класами. Однак до членів типу оголошених в базовому класі як закриті, незважаючи на їх успадкування, похідний клас отримати не може.

Програміст типів вільний заборонити спадкування від класу на будь-якому рівні ієрархії, оголосивши клас запечатаним за допомогою ключового слова **sealed** :

```

class A {}
sealed class B : A {}
class C : B // Викликає помилку компіляції

```

## 2. Спадкування членів класу

Призначення успадкування в C # – це повторне використання коду і безпечне розширення функціональності. Говорячи, що клас A спадкоємець класу B, ми маємо на увазі, що похідний клас (B) успадковує члени типу класу A, але може визначати і власні, а також змінювати реалізацію базових. Не всі члени класу беруть участь у спадкуванні.

### Успадковані члени класу:

- поля;
- методи;
- події;
- індексатори;
- властивості.

### Неуспадковані члени класу:

- конструктори;
- деструктори (фіналізатор);
- статичні члени.

## 2.1 Спадкування полів

Успадковуються будь-які поля крім статичних.

```

class Drink
{

```

```

    // Оголошений відкритим нестатичні поля - будуть успадковані
    похідними класами
    public string Description;
    public double Price;
    // Оголошується статичне поле, не може буде успадкованим
    public static int YearofPack;
    // Далі мають бути конструктори, властивості та методи
}
class SoftDrink : Drink // Клас SoftDrink похідний від Drink
{
    // Тут неявно присутні поля, успадковані від базового класу
    public string nameofdrinkclass; // Новое поле
    // Далі повинні бути визначені та перевизначені властивості та
    методи похідного класу
}

```

Компілятор розпізнає до яких саме полів (успадкованим або базовим) звертається програміст по об'єкту використаного типу:

```

Drink dr1 = new Drink();
// Звернення до поля базового класу
dr1.Description = "Солона рідина";
SoftDrink sd1 = new SoftDrink();
sd1.Description = "Безалкогольний напій з терпким смаком";

```

При визначенні полів в базовому класі корисно відразу вирішити, як буде організований до них доступ в методах та властивостях похідного. Так, наприклад, якщо одне або кілька полів базового класу програміст оголосить закритим (private), то доступ до них з похідного класу (його методів та властивостей) буде неможливий, хоча він їх успадкує.

```

class Drink
{
    private string Description;
    private double Price;
    // Тут мають бути конструктори, властивості та методи
}
class SoftDrink : Drink
{
    // Визначаємо конструктор похідного класу
    public SoftDrink()
    {
        // Наступні інструкції не будуть виконані, оскільки конструктор
        похідного класу не має доступу до полів базового через їх
        модифікатор. Компілятор видає помилку
        Description = "Якись опис";
        Price = 15;
    }
}

```

```
}
```

## 2.2 Спадкування методів

Методи несуть в собі основну функціональність класів. Всі методи крім статичних успадковуються. На них покладається завдання щодо спеціалізації інструкцій або розширення функціоналу базового класу.

Наприклад, для прикладної задачі - скласти ієрархію напоїв в барі. Знадобиться кілька класів, які будуть успадковуватися від одного спільного.

```
class Drink
{
    public string Description;
    public double Price;
    // Визначимо метод
    public double GetPrice()
    {
        return Price;
    }
}
class SoftDrink : Drink // Клас SoftDrink похідний від Drink
{
    // тут неявно присутні поля, успадковані від базового класу
    public string nameofdrinkclass; // Нове поле
    // тут неявно присутня реалізація методу GetPrice()
}
```

Зараз метод `GetPrice ()` можна викликати як з базового, так і з похідного класу. Робить він одне і теж.

```
Drink dr1 = new Drink();
Console.WriteLine (dr1.GetPrice());
SoftDrink sd1 = new Drink();
Console.WriteLine (sd1.GetPrice());
```

Але, припустимо, вартість кожного напою повинна розраховуватися за власною формулою, описані у відповідному методі базового і похідних класів будуть відрізнятися. Такий функціонал доступний через зв'язку однойменних: віртуальних методів базового класу і перевизначених методів похідних класів.

```
class Drink
{
    public string Description;
    public double Price;
    // Визначаємо метод, який може бути перевизначений у похідних
    класах
}
```

```

public virtual double GetPrice()
{
    return Price;
}
}
class SoftDrink : Drink // Клас SoftDrink похідний від Drink
{
    // Тут неявно присутні поля, успадковані від батьківського класу
    public string nameofdrinkclass; // Нове поле
    // Перевизначаємо метод батьківського класу
    public override double GetPrice(double addtoPrice)
    {
        // Інша реалізація методу батьківського класу
        return Price+addtoPrice;
    }
}

```

У клієнтському коді компілятор визначить викликається метод по сигнатурі:

```

Drink dr1 = new Drink();
Console.WriteLine (dr1.GetPrice());
SoftDrink sd1 = new Drink();
Console.WriteLine (sd1.GetPrice(12.08));

```

У керівництві з програмування бібліотеки MSDN від творців C# - Microsoft, рекомендується **всі методи базового класу призначати віртуальними**, так як висока ймовірність що їх реалізацію буде потрібно змінити в похідних класах.

**Віртуальні методи базового класу не обов'язково перевизначати**, допустимо залишати базову реалізацію. Однак, якщо базовий клас абстрактний, тобто його методи не мають реалізації, то успадковані похідними класами методи також не матимуть реалізації. Тому похідний клас обов'язково повинен їх визначати.

Умови доступу до методів базового класу ідентичні умовам доступу до полів. Методи, оголошені в базовому класі як `private` (закриті) будуть успадковані похідному класі, але до них не буде доступу.

## 2.3 Спадкування властивостей

Оскільки властивості об'єднують в собі функції полів і методів, всі умови їх успадкування ідентичні загальних умов успадкування методів і полів:

- властивості можуть бути віртуальними і їх можна перевизначати в похідних класах;
- властивості можуть бути абстрактними, і їх потрібно визначати в похідних класах;

- успадковуються будь-які (автоматичні і неавтоматичні) властивості, крім статичних.

Рекомендований стиль програмування від Microsoft такий:

- Поля базового класу оголошувати захищеними (protected).
- Доступ до захищених полів організовувати через відкриті властивості.
- Якщо повернення або зміна поля в похідному класі повинно відрізнятися від базових, то рекомендується властивості батьківського класу оголошувати віртуальними і перевизначати їх в дочірніх класах.

Для прикладу візьмемо той знайомий клас Drink, злегка змінивши його:

```
abstract class Drink
{
    // Захищені абстрактні поля базового класу
    protected string description;
    protected double price;
    // Оголошуємо, але не реалізовуємо властивість. Вона
    абстрактна, оскільки оголошена в абстрактному класі
    public string Description
    {
    }
}

class SoftDrink : Drink // Клас SoftDrink похідний від Drink.
{
    // Оголошуємо властивість базового класу
    public override string Description
    {
        get {return description;}
        set {description = value;}
    }
    // Оголошуємо автоматичну властивість з неявним визначенням
    нового поля
    public int Nameofdrinkclass {get;set;} // Визначається поле,
    доступ до якого здійснюється тільки через автоматичну властивість
    Nameofdrinkclass
}
```

### 3 Конструктори батьківських та похідних класів

C # не допускає спадкування конструкторів від батьківського класу. Будь-яке побудова об'єкта дочірнього класу пов'язане з викликом конструктора батьківського. Це так звана поетапна збірка.

Оскільки похідний клас - це спеціалізація базового, то багатоетапна логіка побудови заснована на конкретизації об'єкта, кожен конструктор виконує лише

частину загальної логіки побудови: базовий створює базову частину (поля і методи батьківського класу), похідний - спеціалізовану.

Якщо програміст в базовому класі не визначив свій конструктор, то компілятор викликає конструктор за замовчуванням. Але, якщо програмісту потрібно, щоб компілятор викликав конкретний, перевизначений їм конструктор з базового, то він повинен це явно вказати в дочірньому класі, застосувавши ключове слово **base** (посилання на об'єкт базового класу).

### 3.1 Виклик конструктора базового класу

Сигнатура виклику конструктора батьківського класу схожа на звичайне визначення конструктора, тільки злегка розширена:

модифікатор доступу `_ім'я конструктора похідного класу(параметри) : base (параметри) {}`

Коли компілятор зустрічає в коді ключове слово **base**, він вставляє на це місце посилання на поточний об'єкт базового класу.

Через посилання **base** програміст може отримати доступ до будь-якого члену класу. Але тільки з блоку реалізації: конструктора, властивості, індексатора або методу.

```
public class Programmer
{
    protected string phonenumber;
    protected string name;

    public virtual void Info()
    {
        Console.WriteLine("Ім'я: {0}", name);
        Console.WriteLine("Серійний номер: {0}", phonenumber);
    }
    public Programmer(string pn, string n)
    {
        phonenumber = pn;
        name = n;
    }
}

class ClassProgrammer : Programmer
{
    public string idnumber;
    public override void Info()
    {
        // Виклик методу Info() базового класу
        base.Info();
    }
}
```

```

        Console.WriteLine("Ідентифікаційний номер програміста
класів: {0}", idnumber);
    }
    // Конструктор користувача похідного класу з викликом
конструктора батьківського
    public ClassProgrammer(string id, string pn, string n) :
base(pn, n)
    {
        idnumber = id;
    }
}

```

Тепер, коли ми налаштували виклик методів і конструкторів батьківського класу через похідний, використовуємо це при створенні об'єкта:

```

class SomeTestClass
{
    static void Main()
    {
        ClassProgrammer cp2 = new ClassProgrammer("5456658", "22-
6-04", "Роман Данілов");
        /* Викликається спочатку конструктор користувача базового класу,
далі конструктор користувача похідного класу*/

        cp2.Info();
        // Викликаємо спочатку метод базового класу, далі похідного
    }
}

```

### 3.2 Ключове слово this

При найменуванні внутрішніх полів і параметрів часто виникає ситуація, коли в базовому класі та похідних класах різні члени різних класів мають одні й ті ж імена. Для середовища виконання не існує імен, вона працює з об'єктами і посиланнями на них. А ось компілятор намагається розібратися до члену-якого об'єкта забажав отримати доступ програміст.

Наприклад, в наступному прикладі компілятору буде важко розібратися:

```

public class Programmer
{
    string name; // Відкрите поле базового класу
}

public class ClassProgrammer : Programmer
{
    string name; // Інше відкрите поле похідного класу
}

```

При використанні поля з ім'ям `name`, наприклад, в конструкторі компілятор не зрозуміє яке саме поле потрібно форматувати: поле базового класу або похідного. Для конкретизації посилання на потрібний член класу використовується **this**, змінна зберігає посилання на поточний об'єкт.

```
public class ClassProgrammer : Programmer
{
    string name; // Інше відкрите поле похідного класу
    public ClassProgrammer(string name)
    {
        this.name = name; // Ініціалізація поля похідного класу
    }
}
```

Посилання на об'єкт, що зберігається в **this** поміщається туди під час виконання. І там завжди зберігається посилання на поточний об'єкт. Рекомендується з конструкторів та методів дочірніх класів звертатися до своїх членів через **this**, а до членів батьківського типу - через **base**.

Для прикладу розглянемо базовий клас двовимірних графічних об'єктів:

```
public class Object2D {
    public int x, y; //поля-координати
    //конструктор
    public Object2D(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //метод для "відображення" public
    void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}", x, y);
    }
    //метод для "переміщення"
    public void Move(int dx, int dy) { x +=
        dx;
        y += dy; Draw();
    }
}
```

маючи цей клас, не важко створити нові - точку та коло.

```
// клас точка
public class Point : Object2D { public
    Point(int x, int y)
        // конструктор Point посилається
        // на батьківський конструктор - Object2D
        : base(x, y) {
```

```

        //тут може бути додатковий код
    }
}

```

тут ключове слово **base** означає, що при створенні об'єкта точка спочатку викликається базовий конструктор `Object2D`, після чого в середині конструктора `Point` можна виконати унікальні для цього класу дії. Що наочніше демонструється в класі – коло.

```

// клас коло
public class Circle : Object2D {
    public int r; //нове поле - радіус
    public Circle(int x, int y, int r): base(x, y)
    {
        this.r = r;
    }
}

```

В цьому класі введено додаткове поле для радіусу - `r`, і значення поля радіус – задається у власному конструкторі `Circle` після виконання батьківського конструктора `Object2D`.

Обидва ці класи автоматично успадковують методи `Draw()` і `Move(int dx, int dy)` батьківського класу, і результат роботи цих методів ні чим не відрізнятиметься від визову аналогічних методів для об'єкту `Object2D`.

Проте більш цікавим є випадок коли методи відображення (`Draw()`) для кола та точки відрізнятимуться від базових. При цьому при оголошенні батьківського метода повинно використовуватись ключове слово **virtual** (можливий), а при оголошенні методу для дочірнього класу ключове слово – **override** (відхиляти, перевизначити).

Наведемо повний приклад:

```

using System;
using System.Collections.Generic;
using System.Text;
public class Object2D {
    public int x, y; //поля-координати
    //конструктор
    public Object2D(int x, int y) { this.x = x;
        this.y = y;
    }
    //метод для "відображення" public
    virtual void Draw() {
        Console.WriteLine("Об'єкт з координатами {0},{1}", x, y);
    }
    //метод для "переміщення"
    public void Move(int dx, int dy) { x += dx;
        y += dy; Draw();
    }
}

```

```

    }
}

// клас точка
public class Point : Object2D { public Point(int
    x, int y)
    // конструктор Point посилається
    // на батьківський
    // конструктор - Object2D
    : base(x, y) {
    }

    public override void Draw() {
        //повністю перебиває
        //батьківський метод Console.WriteLine(
            "Точка з координатами {0},{1}",x, y);
    }
}

// клас коло
public class Circle : Object2D {
    public int r; //нове поле - радіус
    public Circle(int x, int y, int r) : base(x, y)
    {
        this.r = r;}
    public override void Draw() {
        //спочатку викликає батьківський метод
        base.Draw();
        //потім доповнює його
        Console.WriteLine("це коло радіусом {0}", r);
    }
}

class Test_70 {
    static void Main() {
        Point p0 = new Point(10,20); Point
        p1 = new Point(1, 40);
        Circle c0 = new Circle(15, 17, 10);
        p0.Draw();
        p1.Draw();
        c0.Draw(); Console.ReadKey();
    }
}

```

Ця програма виведе на екран наступну інформацію:

Точка з координатами 10,20  
 Точка з координатами 1,40  
 Об'єкт з координатами 15,17  
 це коло радіусом 10

Крім того, з усіма об'єктами нащадками базового класу можна спільно

працювати, як з об'єктами базового класу. Для прикладу залишимо класи `Object2D`, `Point` та `Circle` без змін перепишемо тільки клас `Test_70`:

```
class Test_71 {  
  
    static void Main() {  
  
        Point p0 = new Point(10,20); Point  
        p1 = new Point(1, 40);  
        Circle c0 = new Circle(15, 17, 10);  
  
        //створення масиву вказівників  
        //на об'єкти Object2D  
        Object2D[] objects = new Object2D[3];  
  
        //спочатку всі елементи цього масиву рівні null  
        //заповнимо його посиланнями на  
        //створені об'єкти  
  
        objects[0] = p0;  
        objects[1] = p1;  
        objects[2] = c0;  
  
        //тепер над ними можна виконувати групові дії  
        //відображення  
        foreach(Object2D o in objects) { o.Draw();  
        }  
        Console.WriteLine("\n\rпісля зміщення\n\r");  
  
        //зміщення  
        foreach(Object2D o in objects) { o.Move(5, -  
            4);  
        }  
        Console.ReadKey();  
    }  
}
```

Результатом роботи цієї програми буде:

Точка з координатами 10,20  
Точка з координатами 1,40  
Об'єкт з координатами 15,17  
це коло радіусом 10

після зміщення

Точка з координатами 15,16  
Точка з координатами 6,36  
Об'єкт з координатами 20,13

це коло радіусом 10

### 3.3 Індексатор

Крім вищезгаданих операторів можна перевантажити оператор [], але для цього використовують спеціальну властивість – *індексатор*. Індексатор можна використовувати для доступу до різних полів об'єкта або полів-масивів, оголошуються вони за допомогою ключового слова `this` та квадратних дужок [].

Приклад використання індексатору для різних полів:

```
using System;

public class Car {                                     //клас машина

    public string color; //колір
    public string model; //модель
    public int yearBuilt; //рік випуску

    // індексатор
    public string this[int i] {
        // властивість на читання get {
            switch(i) {
                case 0:
                    return model; case
                1:
                    return color; case
                2:
                    return yearBuilt.ToString(); default:
                    return null;
            }
        }
        // властивість для запису set {
            switch(i) {
                case 0:
                    model=value; break;
                case 1:
                    color=value; break;
                case 2:
                    yearBuilt = Convert.ToInt32(value); break;
                default:
                    break;
            }
        }
    }

}

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

```

        Car c = new Car();

        c[0] = "Audi";
        c[1] = "Green";
        c[2] = "2000";
        for(int i = 0; i < 3; i++) { Console.WriteLine(c[i]);
        }
        Console.ReadKey();
    }
}

```

На екрані буде наступне:

```

Audi
Green
2000

```

В цій програмі використання індексатора дозволяє поставити у відповідність кожному полю класа індекс і звертатися до полів по імені об'єкта за допомогою цього індексу.

Приклад використання індексатору для масиву:

```

using System;
public class Vector { int[] v;

    // конструктор
    public Vector(int size) { v =
        new int[size];
    }

    // метод
    public int Length() { if(v
        == null) {
            return 0;
        } else {
            return v.Length;
        }
    }

    // індексатор
    public int this[int i] { get {
        return v[i]; }
        set { v[i] = value; }
    }
}

class Program {
    static void Main(string[] args) {
        Vector vect = new Vector(5);
        for(int i = 0; i < vect.Length(); i++) { vect[i] = i *

```

```

        2;
        Console.WriteLine(
            "vect[{0}] = {1}", i, vect[i]
        );
    }
    Console.ReadKey();
}
}

```

Програма виведе на екран:

```

vect[0] = 0
vect[1] = 2
vect[2] = 4
vect[3] = 6
vect[4] = 8

```

Як бачимо з цього прикладу, використання індексатору дозволяє звертатися до елементів поля масиву `v`, як до елементів об'єкту `vect`.

<https://brainoteka.com/blogs/c-spravochnik/nasledovanie>

<https://metanit.com/sharp/tutorial/3.7.php>