

КЛАСИ В С#. ПОЛЯ ТА МЕТОДИ

Клас – це шаблон який визначає форму, зміст та поведінку об'єкту. **Об'єкт** – це екземпляри класу. За допомогою класу реалізується перший з основних принципів ООП – **інкапсуляція**. Інкапсуляція – це об'єднання в одному цілому даних та алгоритмів обробки цих даних. Інкапсуляція – це приховування від зовнішнього користувача деталей реалізації об'єкта. Дані зберігаються у вигляді **полів**, а алгоритми обробки у вигляді **методів**. Поля, ще іноді називають змінними класу. Оголошення класу в мові С# наступне:

```
[модифікатор_доступу] class <ім'я класу> {  
    // оголошення полів  
    [мод._доступу] <тип1> <ім'я_поля1>  
        [=початкове_значення1];  
    [мод._доступу] <тип2> <ім'я_поля2>;  
        [=початкове_значення2]  
    // ...  
    // оголошення методів  
    [мод._доступу] <тип_значення1> <ім'я_методу1>  
        ([параметри1]){  
        //тіло методу  
    }  
    [мод._доступу] <тип_значення2> <ім'я_методу2>  
        ([параметри2]){  
        //тіло методу  
    }  
    //...  
}
```

МОДИФІКАТОРИ РІВНЯ ДОСТУПУ

Модифікатор	Опис
private	компонент доступний тільки в середині класу, приймається за замовчуванням
protected	компонент доступний тільки в середині класу або в класі нащадку
internal	компонент доступний тільки в середині класу або в цій же програмі (або зборці)
protected internal	компонент доступний тільки в середині класу, в класі нащадку або в цій же програмі (або зборці)
public	компонент доступний без обмежень

Приклад класу, який складається тільки з полів:

```
public class Transformer {    //клас трансформатор
    public int windingCount = 2; //кількість обмоток
    public int temperature;    //поточна температура
    public string power;      //потужність
    public string model;      //марка
    public int yearBuilt;     //рік випуску
}
```

МЕТОДИ КЛАСУ

Якщо метод нічого не повертає – цей тип задається ключовим словом **void**, інакше тіло циклу обов'язково повинно містити оператор **return**, за допомогою якого метод повертає значення заданого типу. Якщо тип об'єктний, то повертається посилання (вказівник) на цей об'єкт. Параметри1, параметри2 – необов'язковий список параметрів вигляду:

```
[ref|out|params] <тип_параметру1> <ім'я_параметру1>
[, <тип_параметру2> <ім'я_параметру2>[, ...]]

public class Power {    //клас електрична потужність
    //... поля якщо потрібно
    // метод обчислює повну потужність
    public float Full(float U, float I) {
        return U * I;
    }
    // метод для обчислення активної потужності
    public float Active(
        float U, float I, float cosFi
    ) {
        return Full(U, I) * cosFi;
    }
}
```

СТВОРЕННЯ ОБ'ЄКТУ. КОНСТРУКТОР КЛАСУ

Для описаного класу `Transformer` об'єкт створюємо за допомогою ключового слова **new**.

```
Transformer myTrans1 = new Transformer();
```

Тепер можемо звертатися до полів та методів створеного об'єкту використовуючи оператор «крапка» (.) разом з посиланням на об'єкт:

```
myTrans1.windingCount = 2;  
myTrans1.model = "АТДЦТН-400/330/110/35";  
myTrans1.yearBuilt = 2003;  
Console.WriteLine("Марка={0}", myTrans1.model);
```

У цьому прикладі `Transformer()` після ключового слова **new** — це *конструктор класу*. Це спеціальний метод, який викликається при створенні об'єкту і ім'я його співпадає з іменем класу.

СТВОРЕННЯ ОБ'ЄКТУ. КОНСТРУКТОР КЛАСУ

```
1 class Person
2 {
3     public string name;
4     public int age;
5
6     public Person() { name = "Неизвестно"; age = 18; }    // 1 конструктор
7
8     public Person(string n) { name = n; age = 18; }        // 2 конструктор
9
10    public Person(string n, int a) { name = n; age = a; }   // 3 конструктор
11
12    public void GetInfo()
13    {
14        Console.WriteLine($"Имя: {name}  Возраст: {age}");
15    }
16 }
```

```
1 static void Main(string[] args)
2 {
3     Person tom = new Person();           // вызов 1-ого конструктора без параметров
4     Person bob = new Person("Bob");      // вызов 2-ого конструктора с одним параметром
5     Person sam = new Person("Sam", 25);  // вызов 3-его конструктора с двумя параметрами
6
7
8     bob.GetInfo();                       // Имя: Bob  Возраст: 18
9     tom.GetInfo();                       // Имя: Неизвестно  Возраст: 18
10    sam.GetInfo();                       // Имя: Sam  Возраст: 25
11 }
```

Ім'я: Невідомо Вік: 18

Ім'я: Bob Вік: 18

Ім'я: Sam Вік: 25

КЛЮЧОВЕ СЛОВО **THIS**

Ключове слово **this** є посилання на поточний екземпляр класу. Можемо не дублювати функціональність конструкторів, а просто звертатися з одного конструктора до іншого через ключове слово **this**, передаючи потрібні значення для параметрів:

```
1 class Person
2 {
3     public string name;
4     public int age;
5
6     public Person() : this("Неизвестно")
7     {
8     }
9     public Person(string name) : this(name, 18)
10    {
11    }
12    public Person(string name, int age)
13    {
14        this.name = name;
15        this.age = age;
16    }
17    public void GetInfo()
18    {
19        Console.WriteLine($"Имя: {name}  Возраст: {age}");
20    }
21 }
```

```
1 Person tom = new Person { name = "Tom", age=31 };
2 tom.GetInfo();           // Имя: Том  Возраст: 31
```

Щоб розмежувати параметри і поля класу, до полів класу звернення йде через ключове слово **this**. Так, у виразі **this.name = name**; перша частина **this.name** означає, що **name** - це поле поточного класу, а не назва параметру **name**.

Для ініціалізації об'єктів класів можна застосовувати **ініціалізатор**. Ініціалізатори представляють передачу в фігурних дужках значень доступним полям і властивостями об'єкта

ТИПИ ПЕРЕДАЧІ ПАРАМЕТРІВ

```
using System;  
using System.Collections.Generic;  
using System.Text;
```

```
class Test_ref {  
    void Mov(int a, int b) {  
        int x = a;  
        a = b;  
        b = x;  
    }  
    void Xchg(ref int a, ref int b) {  
        int x = a;  
        a = b;  
        b = x;  
    }  
    static void Main(string[] args) {  
        int a = 3, b = 4;  
        Test_ref p = new Test_ref();  
        p.Mov(a, b);  
        //значення a, b не змінюються  
        Console.WriteLine("a={0}, b={1}", a, b);  
        p.Xchg(ref a, ref b);  
        //a та b обмінялися значеннями  
        Console.WriteLine("a={0}, b={1}", a, b);  
        Console.ReadKey();  
    }  
}
```

передача за значенням

передача за посиланням

a=3, b=4
a=4, b=3

ТИПИ ПЕРЕДАЧІ ПАРАМЕТРІВ

```
using System;
using System.Collections.Generic;
using System.Text;

class Test_out {
    void Power(int c, out int a, out int b) {
        a = c * c;
        b = c * c * c;
    }
    static void Main(string[] args) {
        int a, b, c = 5;
        Test_out p = new Test_out();
        p.Power(c, out a, out b);
        Console.WriteLine("a={0}, b={1}", a, b);
        Console.ReadKey();
    }
}
```

Виведе на екран значення квадрату та кубу числа 5:

a=25, b=125

ТИПИ ПЕРЕДАЧІ ПАРАМЕТРІВ

```
using System;
using System.Collections.Generic;
using System.Text;

class Test_params {
    int Sum(params int[] a) {
        int s = 0;
        foreach(int i in a) {
            s += i;
        }
        return s;
    }

    static void Main(string[] args) {
        Test_params p = new Test_params();
        Console.WriteLine(
            "Sum={0}", p.Sum(1, 2, 3, 4)
        );
        Console.WriteLine(
            "Sum={0}", p.Sum(1, 2, 3, 4, 5)
        );
        Console.ReadKey();
    }
}
```

Sum=10

Sum=15

ПЕРЕВАНТАЖЕННЯ МЕТОДІВ

Перевантаження методів дозволяє використовувати одні й ті самі імена методів змінюючи набір аргументів. Це корисно коли схожі дії треба виконати для різних типів даних, або для різних наборів даних.

```
Sum=2
Sum=7, 8
Sum=8
```

```
using System;
using System.Collections.Generic;
using System.Text;

class Test_Overload {
    //для додавання цілих
    int Add(int a, int b) {
        return a + b;
    }
    //для додавання дійсних
    double Add(double a, double b) {
        return a + b;
    }
    //для збільшення на 1
    int Add(int a) {
        return ++a;
    }
    static void Main(string[] args) {
        Test_Overload t = new Test_Overload ();
        Console.WriteLine("Sum={0}", t.Add(1, 2));
        Console.WriteLine("Sum={0}", t.Add(3.4, 4.4));
        Console.WriteLine("Sum={0}", t.Add(7));
        Console.ReadKey();
    }
}
```