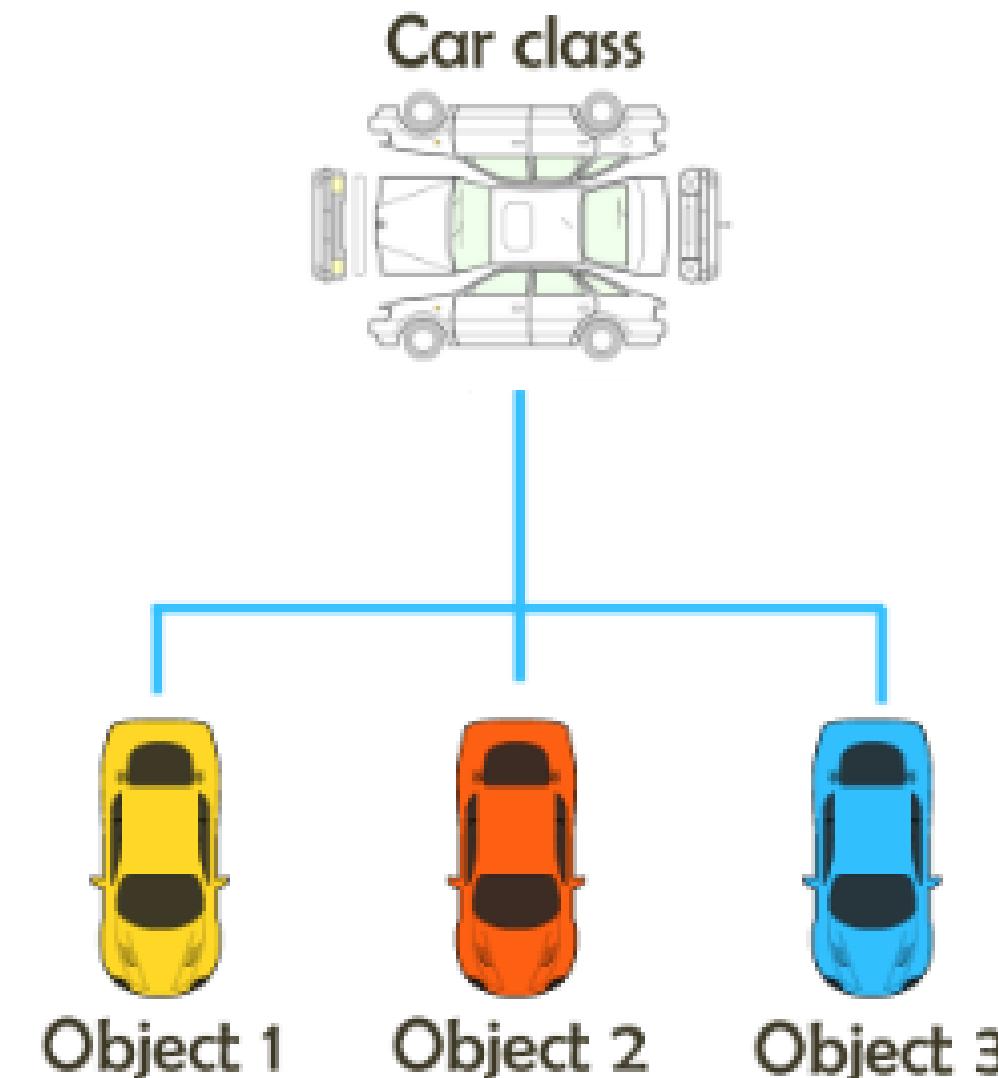




ASICODE

Курс Основи - C#

ЗАНЯТТЯ № 8 - КЛАСИ ТА ОБ'ЄКТИ.
ПОЧАТКОВЕ ЗНАЙОМСТВО З ОВП



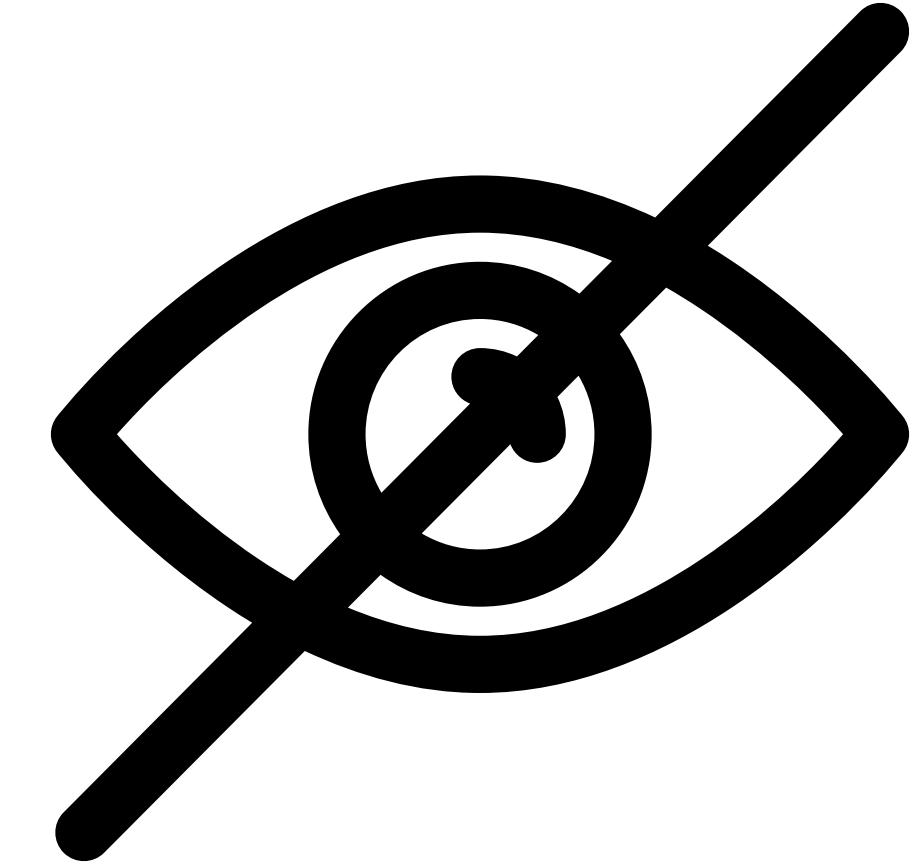
Основи Об'єктно-орієнтованого програмування

Об'єктно-орієнтоване програмування (ООП) є однією з найбільш популярних парадигм у світі промислової розробки програмного забезпечення. Основними будівельними елементами ОВП є класи та об'єкти. Для інтуїтивного розуміння цих понять наведемо такий приклад: аналогом класу у житті є креслення будівлі чи автомобіля, тобто, деякий шаблон. Об'єкт - це безпосередньо реалізація класу у вигляді деякої сутності, в нашій аналогії - це конкретна будівля або конкретний автомобіль, виконаний за кресленням.

Виділяють три основні "стовпи" ООП - це інкапсуляція, успадкування і поліморфізм.



ASICODE



Інкапсуляція

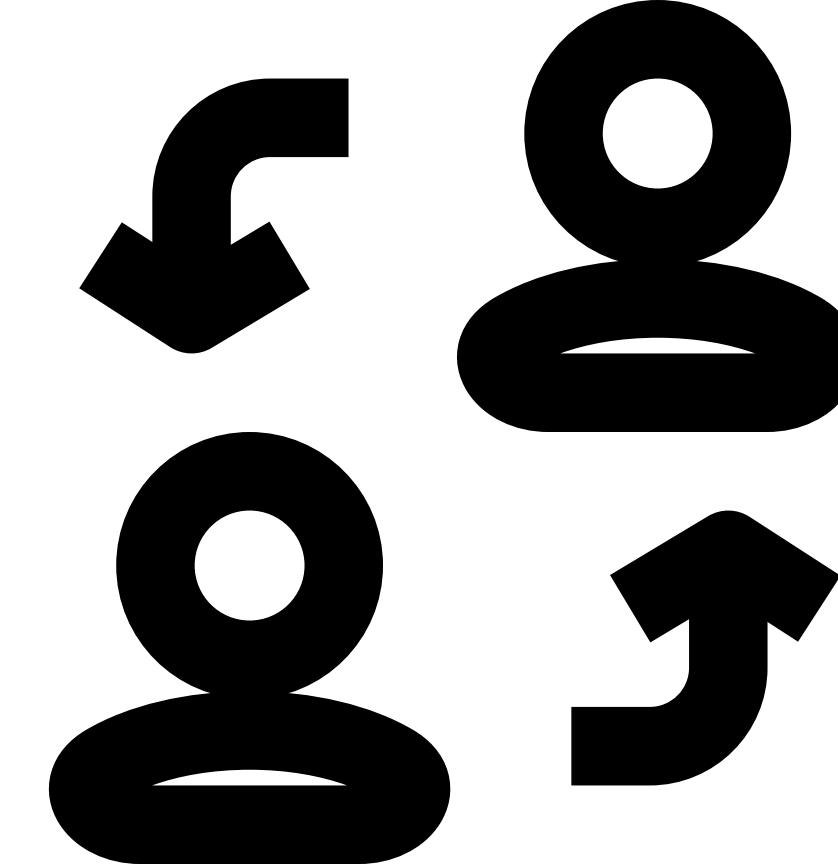
Інкапсуляція передбачає два основні моменти:

- приховування деталей реалізації;
- зв'язування даних та методів для їх обробки в одному місці.



успадкування

Спадкування – це концепція, яка передбачає, що один клас може успадковувати функції та дані іншого класу. Клас, від якого виробляється успадкування називається батьківським чи базовим класом, клас який успадковує - спадкоємцем . Відношення між класом спадкоємцем та базовим класом можна визначити словом “ε”.



Поліморфізм

Говорячи про поліморфізм загалом, можна сказати, що це можливість обробки даних різних типів однією і тією самою функцією. Розрізняють параметричний поліморфізм та ad – hoc поліморфізм. Параметричний поліморфізм передбачає, що той самий код функції може працювати з даними різних типів. Ad – hoc поліморфізм передбачає створення різних реалізацій функцій в залежності від типу аргументу (ів), при цьому їх сигнатура (без урахування типів вхідних аргументів) залишається однією і тією ж.



Класи



Клас у мові C# оголошується за допомогою ключового слова `class` перед ним можуть стояти кілька модифікаторів, після розташовується ім'я класу. Якщо передбачається, що клас є спадкоємцем іншого класу або реалізує один або кілька інтерфейсів, то вони відокремлюються двокрапкою від імені класу та перераховуються через кому.

```
class Building  
{  
}
```

Усередині себе, клас може містити методи, поля та властивості. Методи схожі функції з мови структурного програмування. Фактично вони визначають те, як можна працювати з цим класом чи об'єктами класу. Поля - це змінні, пов'язані з цим класом, а властивості - це конструкції спеціального виду, які спрощують роботу з полями.

Модифікатори доступу

У C# доступні такі модифікатори доступу: public , private , protected



public

- для класу: вказує на те, що клас доступний зовні;
- для складових класу: вказує, що вони можуть бути доступні поза класом, до них можна отримати доступ з будь-якого місця в програмі.

private

- для класу: вказує на те, що клас доступний лише всередині збирання.
- для складових класу: вказує на те, що вони доступні лише всередині класу.

protected

- для класу: вказує на те, що клас доступний лише всередині збирання;
- для складових класу: вказує на те, що вони доступні лише всередині класу та класів нащадків.

Конструктор класу

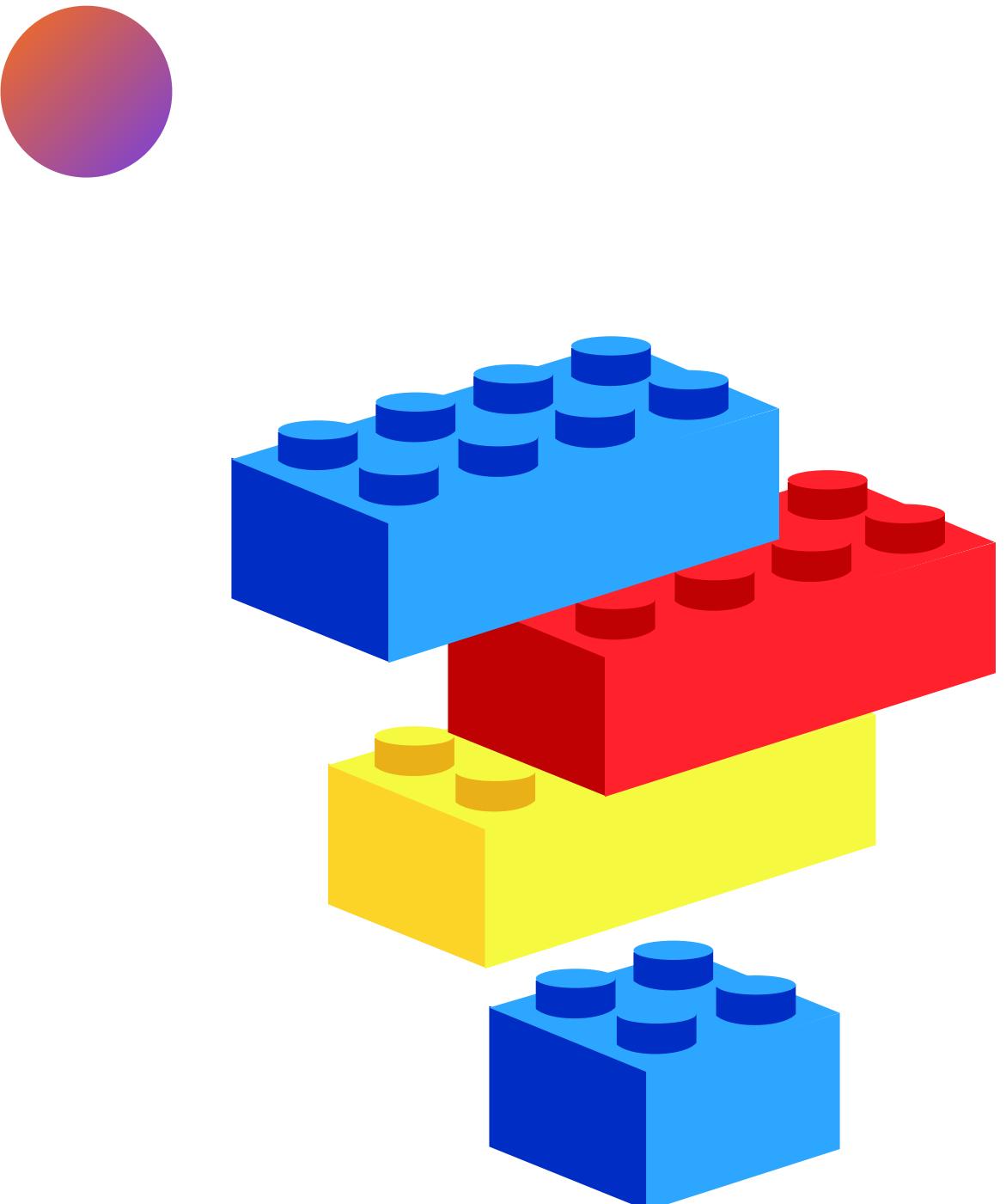
це спеціальний метод, який викликається під час ініціалізації об'єкта за допомогою ключового слова `new`. Ім'я конструктора має збігатися з ім'ям класу, в сигнатурі конструктора не вказується тип значення, що повертається.

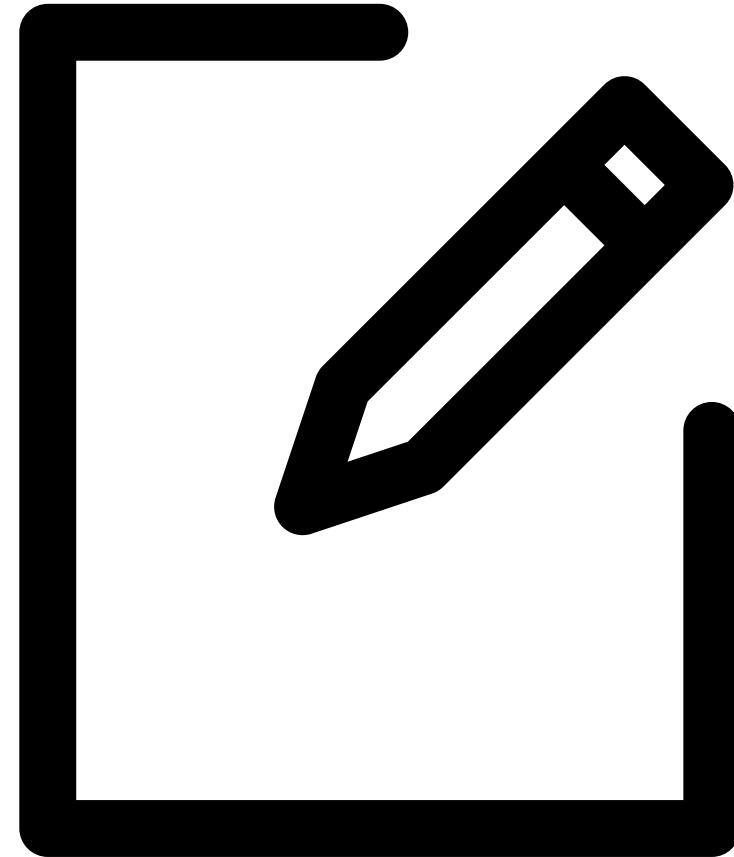
Ось так буде виглядати найпростіший варіант конструктора для класу `DemoClass` :

```
// Конструктор класу
public DemoClass ()
{}
```

Про конструктор класу слід знати наступне:

- Конструктор можна не створювати явно, тоді для класу буде створено за замовчуванням конструктор, він виглядає наступним чином:
- Конструкторів може бути кілька, вони повинні відрізнятися за кількістю і типом аргументів, що приймаються:
- Якщо ви створите конструктор, який містить набір аргументів, конструктор за замовчуванням вже не буде створений для класу. Якщо ви хочете створювати об'єкти без вказівки аргументів, необхідно додати до класу відповідний конструктор.





Ініціалізація об'єктів класу

Ініціалізація об'єкта класу здійснюється за допомогою ключового слова `new`. Створимо об'єкт класу `DemoClass`:

```
DemoClass demo = New DemoClass();
```

Якщо клас має кілька конструкторів, то при ініціалізації можна вибрати один з існуючих:

```
DemoClass d2 = New DemoClass(1);  
d2. Method(); // field: 1, Property: 0  
DemoClass d3 = new DemoClass(1, 2);  
d3. Method(); // field: 1, Property: 2
```

Доступна можливість використання неявного оголошення за допомогою ключового слова `var`:

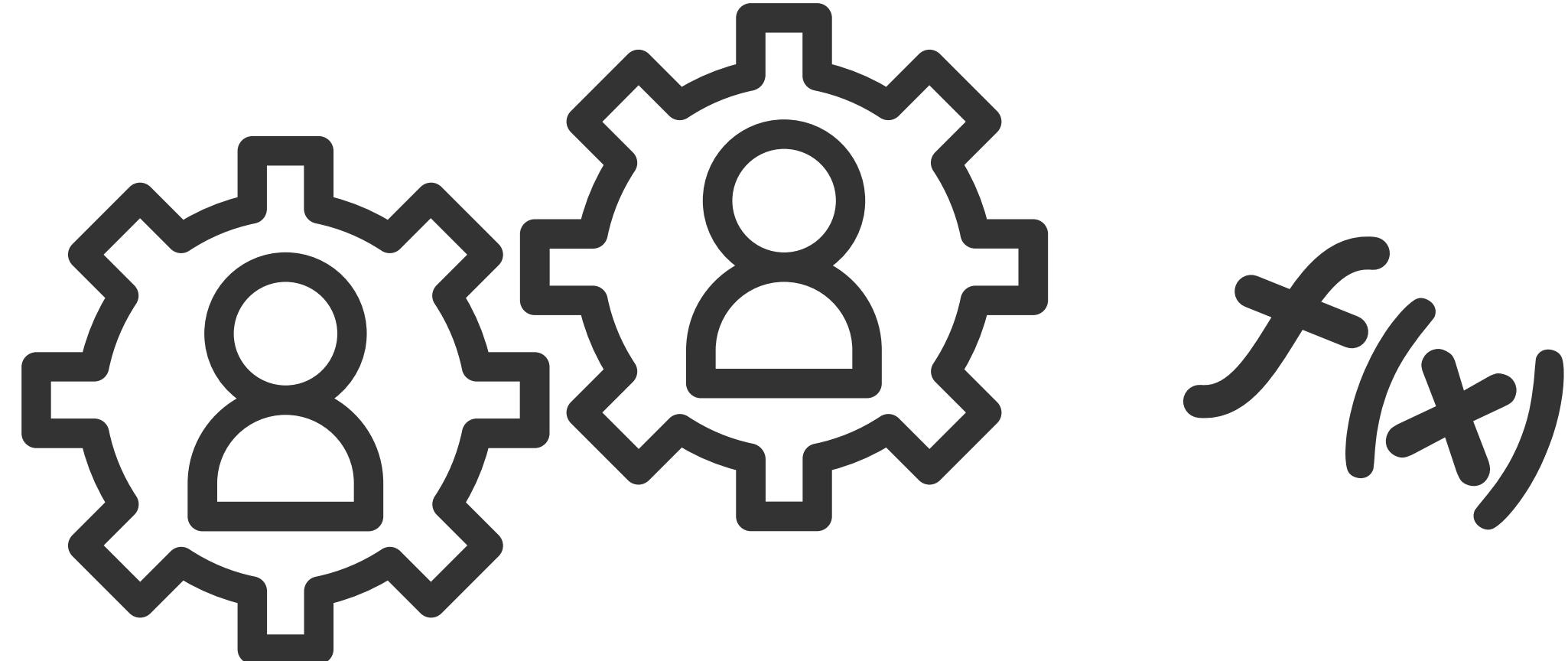
```
var d4 = New DemoClass();
```

Якщо клас має публічні властивості, їм можна присвоїти значення при ініціалізації:

```
var d5 = new DemoClass(10) { Property = 11 };  
d5. Method(); // field: 10, Property: 11
```



ASICODE



Методи

Методом класу називають функцію чи процедуру, що належить класу чи об'єкту. Відмінність функції від процедури у цьому, що функція повертає значення, а процедура немає. У загальному вигляді синтаксис оголошення методу виглядає так:

модифікатор(и) тип_повертаного_значення ім'я_функції(аргументи)



ASICODE



Поля

Поле є змінною будь-якого типу, оголошено всередині класу. Через модифікатор доступу можна керувати рівнем доступу до поля (як і для методів), через ключове слово static можна визначати приналежність поля об'єкту або класу:



```
class DemoClass  
{  
    // ...  
    int field = 0 ;  
    public int publicField = 0 ;  
    public static int publicStaticField = 0 ;  
    // ...  
}
```

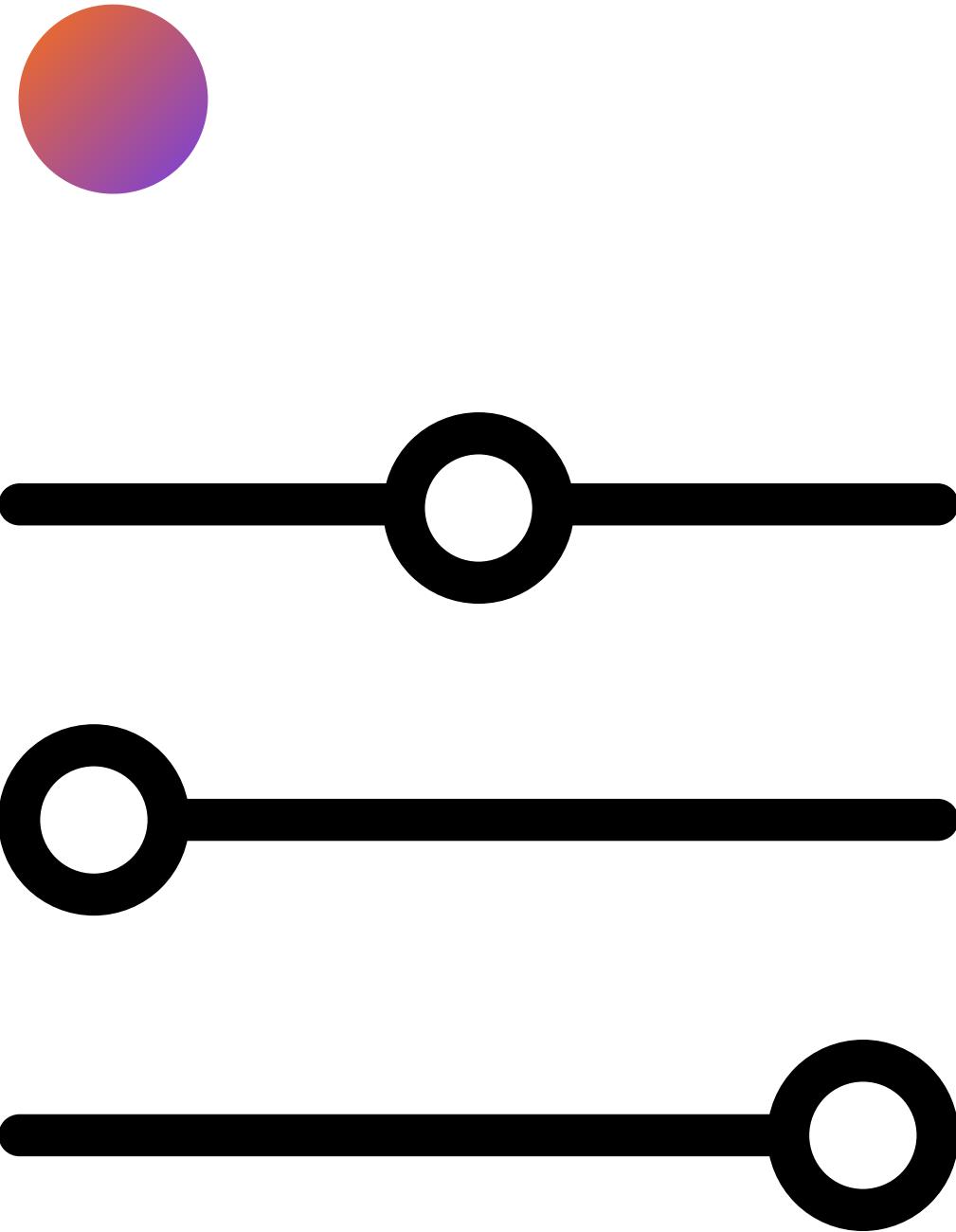


Властивості

Основна причина створення getter'ів та setter'ів - це впровадження додаткової логіки, яка може знадобитися перед тим як видати значення поля або привласнити йому нове значення. Для спрощення процесу вирішення такої задачі мова C# пропонує альтернативу - Властивості . З точки зору користувача класу, робота з властивостями схожа на роботу з відкритими полями. Але всередині, під капотом, це організовано так, що ми можемо додавати логіку при читанні і запису.

Приклад:

```
public float Height  
{  
    get => height;  
    set => height = value;  
}
```





ASICODE

Дякую за увагу