

КЛАСИ ТА ОБ'ЄКТИ

ООП це стиль програмування, який призначений, щоб зробити програмування наближеним до реальний світу.

У програмуванні **об'єкти** є самостійними одиницями, і кожен має свою ідентичність, як і об'єкти в реальному світі.

Об'єкт - яблуко; кружка. У кожного своя унікальна ідентичність. Можливо, є дві кружки, які виглядають однаково, але це все-таки окремі, унікальні об'єкти

Об'єкт може містити інші об'єкти, але вони все ще різні об'єкти. Об'єкти також мають **характеристики**, які використовуються для їх опису. Наприклад, машина може бути червоною або синьою, кружка може бути повною або порожньою. Ці характеристики також називають **атрибутами**. Атрибут описує поточний **стан** об'єкта. Об'єкти можуть мати кілька атрибутів (кружка може бути порожньою, червоною і великою).

Стан об'єкта не залежить від його типу; чашка може бути сповнена води, інша може бути порожньою

У реальному світі кожен об'єкт поводить себе по-своєму. Машина рухається, телефон дзвонить тощо.
Те саме стосується об'єктів - поведінка специфічна для типу об'єкта.

У програмуванні об'єкт є **автономним**, зі своїм власним **ідентифікатором**. Він відокремлений від інших об'єктів. Кожен об'єкт має свої **атрибути**, які описують його поточний стан. Кожен об'єкт демонструє власну **поведінку**, що об'єкт може робити

Person Object	Person Object	Car Object
name: "John" age: 25	name: "Amy" age: 22	color: red year: 2015
talk()	talk()	start() stop() horn()

КЛАС

Об'єкти створюються за допомогою класів. Клас описує, яким буде об'єкт, але є відокремленим від самого об'єкта. Клас це як **план, опис** або **визначення об'єкта**.

Об'єктом називають **екземпляр** класу

- Собака – це клас
- Собака Жучка з 3 під'їзду - це об'єкт, представник або екземпляр класу «Собака»

Спочатку визначаємо клас, який стає планом для створення об'єктів. Кожен клас має **ім'я** та описує **атрибути** та **поведінку**.

У програмуванні термін **тип** використовується для позначення імені класу. Створюється об'єкт певного типу. Атрибути також називаються **властивостями** або **даними**.

МЕТОДИ

Метод - ще один термін поведінки класу. Метод - це в основному функція, що належить до класу. Методи подібні до функцій - це блоки коду, які викликаються, і вони також можуть виконувати дії та повертати значення

Наприклад, якщо ми створюємо банківську програму, ми можемо дати класу такі характеристики:

name (ім'я): BankAccount

attributes (атрибути): accountNumber, balance, dateOpened

behavior (поведінка): open(), close (), deposit ()

Клас визначає, що кожен об'єкт повинен мати визначені атрибути та поведінку. Однак він не вказує, які фактичні дані; він лише дає визначення.

Після того, як написали клас, переходять до створення об'єктів, що базуються на цьому класі. Кожен об'єкт називається **екземпляром** класу. Процес створення об'єктів називається **instantiation**. Кожен об'єкт має свій ідентифікатор, дані та поведінку

МЕТОДИ КЛАСУ

Клас може містити один або більше **методів**, що дозволяють здійснювати маніпуляцію даними об'єкта.

Метод об'єкта - програмний код, виконаний у вигляді функції, що реагує на передачу об'єкту певного повідомлення.

Виклик методу об'єкта може призводити до зміни його стану (значення членів-даних), а може і не приводити.

Приклад 1: Пошук і заміна тексту в документі

Приклад 2: Перевірка правопису тексту

Властивість - складова частина об'єкта, доступ до якої здійснюється програмістом як до змінної об'єкта.

Властивості

*Координати вершини А
Координати вершини В
Координати вершини С
Площа
Периметр
Координати центру
вписаного кола*

Методи

*Перемістити в заданому
напрямку
Масштабувати
Повернути навколо
заданої точки*

ОГОЛОШЕННЯ КЛАСУ

Визначення класу починається з ключового слова **class**. За ключовим словом йде назва класу та його тіло, що поміщається у набір фігурних дужок.

```
class BankAccount {  
};
```

Визначення класу повинно закінчуватись **крапкою з комою**. Всі атрибути та поведінка (або члени) повинні бути визначені в тілі класу, в межах фігурних дужок.

Для членів класу визначають **специфікатор доступу** (**public**, **private**, **protected**). До членів, які були визначені за допомогою ключового слова **public**, можна отримати доступ за межами класу.

Створимо клас з одним публічним методом, який буде виводити на екран "Hi"

```
class BankAccount {  
    public:  
    void sayHi() {  
        cout << "Hi" << endl;  
    }  
};
```

Наступним кроком є створення об'єкта класу **BankAccount** - **test**.

```
int main()  
{  
    BankAccount test;  
    test.sayHi();  
}
```

Роздільник точка (.) використовується для доступу та виклику методу об'єкта.

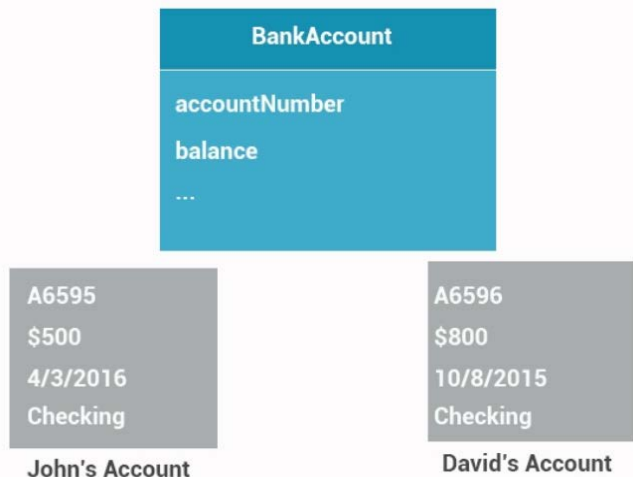
Клас повинний бути оголошений до його використання, як це робимо з функціями

Абстракція

Абстракція даних – це концепція надання зовні тільки необхідну інформацію. Це процес представлення основних особливостей **без включення деталей щодо імплементації**.

Прикладом абстракції з реального світу – це книга. Коли ви чуєте термін книга, ви не знаєте точної специфіки, тобто: кількість сторінок, колір, розмір, але ви розумієте **ідею книги** - абстракцію книги.

Наприклад, коли використовуємо **cout**, фактично використовуєте об'єкт **cout** класу **ostream**. Не потрібно розуміти, як **cout** відображатиме текст на екрані користувача. Єдине, що потрібно знати, як цим користуватися.



Абстракція дозволяє написати один клас банківських рахунків, а потім створити різні об'єкти на основі класу для окремих банківських рахунків, а не створити окремий клас для кожного банківського рахунку.

```

#include <iostream>
#include <string>
using std::string;
using std::cout;
using std::endl;
class Personal
{
public:
    string name;
    int age;
    void move() {
        cout << name << " is moving"<< endl;
    }
};
int main()
{
    Personal person;
    person.name = "Tom";
    person.age = 22;
    cout << "Name: " << person.name << "\tAge: " << person.age << endl;
    person.move();

    return 0;
}

```

Name: Tom Age: 22

Tom is moving

Інкапсуляція

Інкапсуляція - здатність об'єкта приховувати внутрішню будову своїх властивостей та методів.

Згідно з цим принципом, клас повинен розглядатися як «чорний ящик». Зовнішній користувач не знає деталі реалізації об'єкта і працює з ним тільки шляхом наданого об'єктом інтерфейсу.

Наприклад, якщо беремо клас **BankAccount**, ми не хочемо, щоб якась інша частина нашої програми мала доступ та змінювала **balance (баланс)** будь-якого об'єкта без використання методів **deposit ()** або **withdraw ()**.

Ми повинні **приховати** цей атрибут, контролювати доступ до нього, щоб він був доступний лише самим об'єктом.

Таким чином, **баланс** не може бути безпосередньо змінений ззовні об'єкта і доступний лише за допомогою його методів.

СПЕЦИФІКАТОРИ ДОСТУПУ

Специфікатори доступу використовуються для встановлення рівнів доступу для певних членів класу.

Існує три рівня специфікаторів доступу: **public**, **protected**, **private**. **Public** елемент доступний за межами класу, і в будь-якому місці в межах об'єкта класу.

```
#include <iostream>
#include <string>
using namespace std;

class myClass {
    public:
        string name;
};

int main() {
    myClass myObj;
    myObj.name = "SoloLearn";
    cout << myObj.name;
    return 0;
}

//Outputs "SoloLearn"
```

Зверніть увагу на дві крапки (:), яке йде після ключового слова **public**.

Модифікатори доступу повинні бути оголошені лише один раз; декілька членів можуть слідувати за одним модифікатором доступу.

Атрибут **name** є публічним; до нього можна отримати доступ і змінити за межами коду.

Private

Private елемент не може бути доступний, або навіть показаний, поза класом; до нього можна отримати доступ лише всередині класу.

Public функція може використовуватися для доступу до **private** елементів. Наприклад:

```
#include <iostream>
#include <string>
using namespace std;

class myClass {
public:
    void setName(string x) {
        name = x;
    }
private:
    string name;
};

int main() {
    myClass myObj;
    myObj.setName("John");

    return 0;
}
```

Атрибут **name** приватний і не доступний ззовні.

Публічний метод **setName()** використовується для встановлення атрибуту **name**.

Якщо специфікатор доступу не визначений, всі члени класу за замовчуванням встановлюються приватними

```
class myClass {
public:
    void setName(string x) {
        name = x;
    }
    string getName() {
        return name;
    }
private:
    string name;
};
```

Метод **getName** () повертає значення приватного атрибуту *name*.

Інкапсуляція використовується, щоб приховати атрибут **name** від зовнішнього коду. Доступ до атрибуту **name** забезпечений за допомогою публічних методів. Дані класу можна читати та змінювати лише за допомогою цих методів.

```
#include <iostream>
#include <string>
using namespace std;

class myClass {
public:
    void setName(string x) {
        name = x;
    }
    string getName() {
        return name;
    }
private:
    string name;
};

int main() {
    myClass myObj;
    myObj.setName("John");
    cout << myObj.getName();

    return 0;
}

//Outputs "John"
```

КОНСТРУКТОРИ

Конструктори класів - це особливі функції членів класу. Вони виконуються щоразу, коли в цьому класі створюються нові об'єкти. Ім'я конструктора ідентично назві класу. Він не має зворотного типу, навіть типу `void`.

Після створення об'єкта типу *myClass*, конструктор автоматично викликається.

```
class myClass {
public:
    myClass() {
        cout << "Hey";
    }
    void setName(string x) {
        name = x;
    }
    string getName() {
        return name;
    }
private:
    string name;
};

int main() {
    myClass myObj;

    return 0;
}

//Outputs "Hey"
```

Конструктор за замовчуванням не має параметрів. Однак при необхідності параметри можуть бути додані до конструктора. Це дає можливість призначити початкове значення об'єкту під час його створення, як показано у наступному прикладі:

```
class myClass {
public:
    myClass(string nm) {
        setName(nm);
    }
    void setName(string x) {
        name = x;
    }
    string getName() {
        return name;
    }
private:
    string name;
};
```

Створюючи об'єкт, тепер потрібно передати параметр конструктора, як це було б під час виклику функції:

```
class myClass {  
    public:  
        myClass(string nm) {  
            setName(nm);  
        }  
        void setName(string x) {  
            name = x;  
        }  
        string getName() {  
            return name;  
        }  
    private:  
        string name;  
};  
  
int main() {  
    myClass ob1("David");  
    myClass ob2("Amy");  
    cout << ob1.getName();  
}  
//Outputs "David"
```

Було визначено 2 об'єкти і використаний конструктор для передачі початкового значення для атрибута **name** для кожного об'єкта.