

# Лекція 2

## Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

# План лекції 2

1

UML діаграма класу

2

Багатофайлові проекти

3

Конструктор класу. Типи конструкторів

4

Деструктор класу

5

Показчик this

Матеріали курсу:

[https://github.com/LRozova/OOP\\_ukr\\_2023](https://github.com/LRozova/OOP_ukr_2023)

# Повторимо!

- ✓ Основні принципи ООП:  
інкапсуляція, поліморфізм, успадкування.
- ✓ Інкапсуляція та абстракція при створенні класу.
- ✓ Клас: дані + методи
- ✓ Режими доступу до елементів класу:  
private, public, protected

# Режими доступу **private** і **public**

- специфікатори доступу **private** і **public** керують видимістю елементів класу.

Елементи, описані після службового слова **private**, видимі тільки всередині класу. Цей вид доступу прийнятий в класі за замовчуванням. Елементи, описані після слова **public**, доступні зовні.

- Інтерфейс класу описується після специфікатора **public**.

Дія будь-якого специфікатора поширюється до наступного специфікатора або до кінця класу. Можна, задавати кілька секцій **private** і **public**, Порядок їх слідування значення не має.

```
class Employee
{private:
    char name[25]; //ім'я
    int age; //вік
    char position[25]; //посада
    int stage; //стаж
public:
    void setName(char *n);
    void setAge(int s) { age = s;}
    int getAge() { return age;}
    int getStage(int currentYear, int
currentMonth, int currentDay);
};
void Employee::setName(char* n)
{
    strcpy(name, n);
}
```

## Опис об'єктів (екземплярів класу)

Employee John;

Employee managers[20];

Employee \*chef;

## Доступ до елементів об'єкта

John.setName(name);

managers[2].setAge(30);

chef->getStage(2020, 2, 18);

**Геттери і сеттери** - методи класу, які організовують доступ до приватних елементів класу, застосовуючи певну логіку доступу, враховуючі небажані ситуації

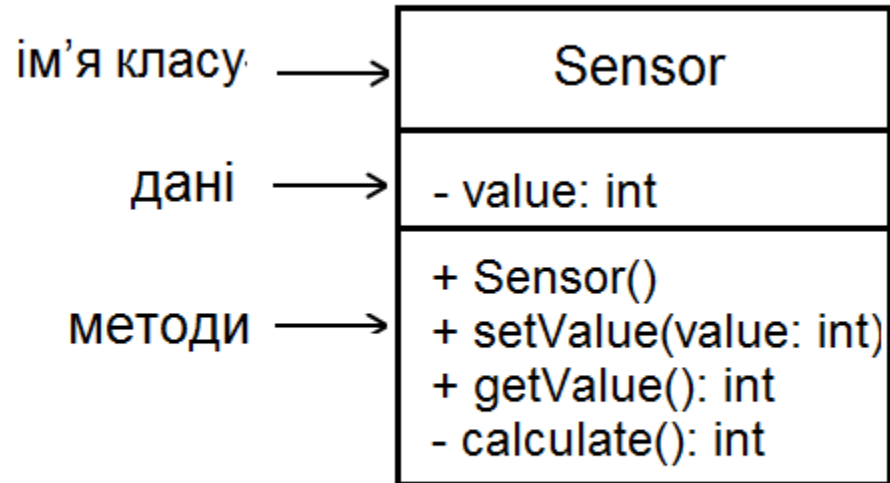
# UML діаграми класів

Одним з способів опису та представлення об'єктно-орієнтованих програм та класів зокрема, є використання **UML-нотації**.

**UML** (*unified modeling language*) — узагальнена мова опису предметних областей, зокрема, програмних систем, об'єктно-орієнтованого аналізу і проектування. Використовується для візуалізації, специфікації, конструювання і документування програмних систем. В контексті ООП в UML містяться такі базові компоненти, як сутності, зв'язки між ними, та набори сутностей

**Структурна сутність класу** уявляє собою опис набору об'єктів з однаковими властивостями (даними або атрибутами), операціями (методами), зв'язками та поведінкою

## UML діаграма класу



public-елементи "+"

private-елементи "-"

protected-елементи "#"

# Відділення інтерфейсу від реалізації

Один з найбільш фундаментальних принципів розробки хорошого програмного забезпечення полягає в відділенні інтерфейсу від реалізації:

при побудові програми на С++ для зручності подальшої підтримки та сприйняття об'єктно-орієнтованого коду програми, слід розділяти оголошення класу та його реалізацію. Для цього оголошення класу слід надавати в заголовкових файлах, а реалізацію — в файлах вихідного коду

**Заголовки - ім'я\_класа.h**

**Файл вихідного коду ( або файл реалізації класу) - ім'я\_класа.cpp**

Заголовки включаються (через #include) в кожен файл, в якому використовується клас, а файли з вихідними кодами компілюються і компонуються з файлом, що містить головну програму (main).



```
//Файл Date.h – заголовковий файл (Header file)
#ifndef DATE_H//умовна компіляція захищає від
#define DATE_H//багаторазового включення файлу

class Date
{
private:
    int m_day;
    int m_month;
    int m_year;
public:
    int getDay(); // геттер для day
    void setDay(int day); // сеттер для day
    int getMonth(); // геттер для month
    void setMonth(int month); // сеттер для month
    int getYear(); // геттер для year
    void setYear(int year); // сеттер для year
};
#endif // DATE_H
//-----Кінець файла-----
```

```
//Файл Date.cpp - файл реалізації
#include <iostream>
#include "Date.h"
using namespace std;
int Date::getDay()
{ return m_day; }
void Date::setDay(int day)
{ m_day = day; }
int Date::getMonth()
{ return m_month; }
void Date::setMonth(int month)
{ m_month = month; }
int Date::getYear()
{ return m_year; }
void Date::setYear(int year)
{ m_year = year; }
//-----Кінець файлу-----
```

```
// Файл main.cpp
```

```
#include <iostream>
```

```
#include "Date.h"
```

```
using namespace std;
```

```
int main()
```

```
{   Date D1;
```

```
    D1.setDay(10);
```

```
    D1.setMonth(11);
```

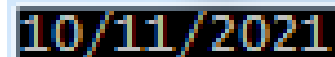
```
    D1.setYear(2021);
```

```
    cout<<D1.getDay()<<"/"<<D1.getMonth()
```

```
    <<"/"<<D1.getYear()<<endl;
```

```
    return 0;
```

```
}
```



10/11/2021

Process returned 0 (0x  
Press any key to conti

# Конструктор класу

**Конструктор** - метод класу, який служить **для створення і ініціалізації екземпляра** класу (об'єкта). Ім'я конструктора завжди збігається з ім'ям класу

- ✓ **Ім'я конструктора – це ім'я класу.** Отже, компілятор відрізняє конструктор від інших методів класу

`Employee ( ) ;`

- ✓ Виконується автоматично в момент створення об'єкта
- ✓ Для створення екземпляра класу потрібно, щоб конструктор був методом типу *public*
- ✓ Конструктор не має типу, не повертає жодного значення, навіть типу `void`
- ✓ **Клас може мати кілька конструкторів** з різними параметрами для різних видів ініціалізації
- ✓ Конструктори не успадковуються

# Деструктор класу

**Деструктор** - метод класу, який служить для видалення екземпляра класу.

- ✓ **Ім'я деструктора** складається з символу **~ (тильда)** і **імені класу**

```
~Employee ();
```

- ✓ Деструктор не має аргументів і значення, що повертається.
- ✓ Деструктор викликається автоматично, коли об'єкт виходить з області видимості:
  - для локальних об'єктів - при виході з блоку, в якому вони оголошені;
  - для глобальних - як частина процедури виходу з main;
  - для об'єктів, заданих через покажчики, деструктор викликається неявно при використанні операції delete.

```
//Файл CreateAndDestroy.h
class CreateAndDestroy
{
public:
    CreateAndDestroy(int); // Конструктор
    ~CreateAndDestroy(); // Деструктор

private:
    int data;
};
//-----Конец файла-----

//Файл CreateAndDestroy.cpp
#include <iostream>
using namespace std;
CreateAndDestroy::CreateAndDestroy(int value)
{
    data = value;
    cout << "Object " << data << " constructor" << endl;
}
CreateAndDestroy::~~CreateAndDestroy()
{
    cout << "Object " << data << " destructor" << endl;
}
//-----Кінець файлу-----
```

```
// Файл main.cpp
```

```
#include <iostream>
```

```
#include "CreateAndDestroy.h"
```

```
using namespace std;
```

```
CreateAndDestroy first(1);
```

```
int main()
```

```
{  cout << "main: Hello" << endl;
```

```
    CreateAndDestroy second(2);
```

```
    cout << "main: second created"<< endl;
```

```
    CreateAndDestroy* third = new CreateAndDestroy(3);
```

```
    cout << "main: third created"<< endl;
```

```
    delete third;
```

```
    cout << "main: third deleted" << endl;
```

```
    cout << "Exit from main" << endl;
```

```
    return 0;
```

```
}
```

```
Object 1 constructor  
main: Hello  
Object 2 constructor  
main: second created  
Object 3 constructor  
main: third created  
Object 3 destructor  
main: third deleted  
Exit from main  
Object 2 destructor  
Object 1 destructor
```

# Типи конструкторів класу

**Конструктор** – метод, який викликається при створенні об'єкта класу

✓ **Конструктор за замовчуванням.** При відсутності в класі будь-якого конструктора створюється компілятором

```
class Person
{
    string Name;
    int Age;
public:
    Person() {cout<<"Створено об'єкт класу Person";}
...
    Person obj1;
```

✓ **Конструктором з параметрами.** Передача початкових значень полям класу

```
Person(string n, int a) {Name = n; Age = a; }
```

У тому числі з параметрами за замовчуванням

```
Person(string n, int a=30)
{ Name = n; Age = a; }
```

```
...
Person obj2("Bill", 20), obj3("Tom");
```



## ✓ Конструктор зі списком ініціалізації

```
Person(string n, int a): Name(n), Age(a)
{ cout<<"Конструктор зі списком ініціалізації"; }
Person obj("Mike", 30);
```

## ✓ Делегуючі конструктори

```
Person(string n)
{ Name = n; }
Person(int a): Person(string n)
{ Age=a; }

Person obj("John"); //викликається перший конструктор
//Person(string n);

Person obj("John",40); //викликається спочатку перший
конструктор, потім другий Person(int a);
```

## ✓ Конструктор копіювання

- Коли один об'єкт класу ініціалізується іншим (створюється копія об'єкту).
- Коли об'єкт передається в будь-яку функцію у якості параметра;
- Коли будь-яка функція повинна повернути об'єкт класу у результаті своєї роботи

class **Dot**

```
{ int *ptr;
```

```
public:
```

```
    Dot() // конструктор за замовчуванням
```

```
{
```

```
    cout << "Звичайний покажчик"<<endl;
```

```
}
```

```
    Dot(const Dot &obj) // конструктор копіювання
```

```
{
```

```
    cout << "Конструктор копіювання"<<endl;
```

```
}
```

```
    ~Dot()
```

```
{ cout << "\nДеструктор\n";}
```

```
};
```

```
void funcShow(Dot object)
```

```
{ cout << “Функція приймає об’єкт класу, як параметр”<<endl;  
}
```

```
Dot funcReturnObject()
```

```
{ Dot object;  
  cout << “Функція повертає об’єкт класу”<<endl;  
  return object;  
}
```

```
int main()
```

```
{  
  Dot obj1; //створюємо об’єкт класу  
  funcShow(obj1); // передаємо об’єкт у якості параметра у функцію  
  funcReturnObject(); //функція повертає об’єкт  
  Dot obj2 = obj1; // ініціалізація об’єкта іншим при створенні  
}
```

Дякую за увагу!