# Лабораторне заняття 3 (2-й семестр)

## Класи та об’єкти. Абстракція та інкапсуляція. Основи UML для представлення класів. Класи в C++

*Мета роботи*: вивчення базових концепцій об’єктно-орієнтованого програмування та основ роботи з класами в C++.

*Завдання на роботу*: опис заданого поняття в вигляді класу з визначенням необхідних атрибутів та методів, представлення його в UML-нотації, програмна реалізація класу.

*Теоретичні відомості до виконання*.

Для моделювання з використанням об’єктно-орієнтованого підходу необхідно:

1. Скласти словник предметної області, в якому визначаються мінімально необхідні поняття (атрибути) для подальшого аналізу. Цей етап має назву абстракція, від його результату залежить уся майбутня робота розроблюваної системи.

2. Визначити, які методи необхідні даному класу для коректної роботи.

3. Визначити, які з атрибутів та методів будуть суто внутрішніми (приватними) для даного класу, а які будуть визначати взаємодію класу з зовнішнім середовищем (іншими класами, програмою взагалі тощо) — публічними. Зазвичай, усі атрибути класу роблять внутрішніми, а для доступу до них та їх зміни створюють спеціальні методи: геттери та сеттери, які є доступними з інших місць програми. Методи класу, навпаки, робляться доступними іншим компонентам програми, винятком можуть бути специфічні методи, необхідні для розрахунків внутрішніх або допоміжних змінних.

4. Визначити типи даних для атрибутів та типи значень, що прийматимуть і повертатимуть методи.

Розглянемо ці етапи на прикладі класу “Робітник компанії”:

1. Етап абстракції. Основними атрибутами класу “Робітник” з точки зору роботодавця, наприклад, можуть бути такі:

- ім’я робітника;
- вік робітника;
- посада;
- заробітна платня робітника;
- кар’єрна історія робітника в цій компанії з назвами посад та датами початку роботи на посадах.

2. Необхідними методами можуть бути такі:

- геттери та сеттери для вказаних в п. 1 атрибутів;
- метод відображення кар’єрної історії;
- метод розрахунку та геттер стажу робітника в компанії;
- метод відображення дати.

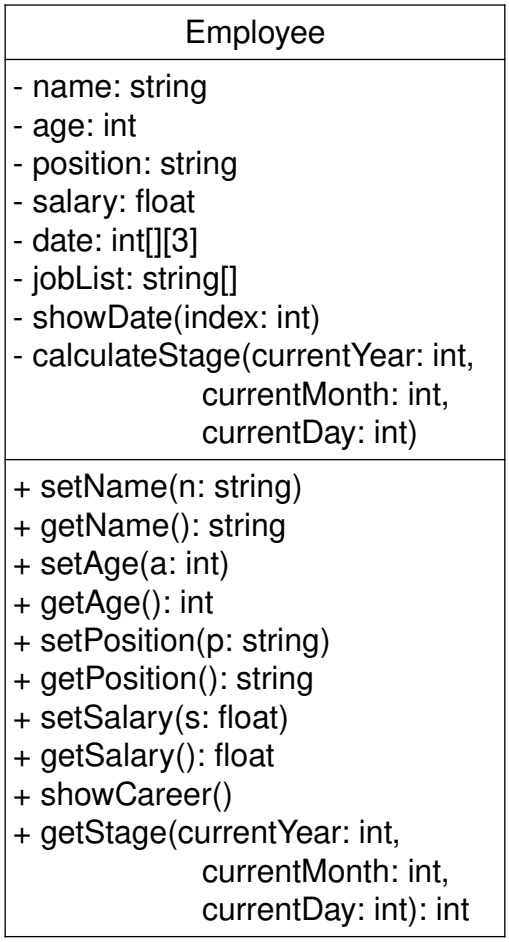
3. Усі атрибути встановлюємо як внутрішні без зовнішнього доступу. Усі геттери, сеттери, методи розрахунку стажу та відображення кар’єрної історії встановлюємо публічними, методи показу дати та розрахунку стажу — приватними.

4. Визначимо типи даних для атрибутів та методів:

- ім’я робітника: **строка**;
- вік робітника: **ціле** число;
- посада: **строка**;

- заробітна платня робітника: **дійсне** число;
- стаж роботи в компанії: **ціле** число;
- кар’єрна історія: **масив строк**;
- дати початку роботи на посадах: двовимірний **масив цілих**;
- геттери повертатимуть змінну відповідного типу і не прийматимуть ніяких параметрів, окрім геттерів ім’я та посади, які прийматимуть посилання на відповідну змінну;
- сеттер посади додатково прийматиме рік, місяць та день зарахування на посаду, при цьому автоматично викликатиметься метод розрахунку стажу;
- метод відображення кар’єрної історії виводить усі елементи масиву посад з відповідними датами;
- метод розрахунку стажу перевіряє чи не дорівнює поточна дата даті останньої посади, якщо вони різні, то перераховується значення стажу, яке потім повертається з методу;
- усі сеттери не повертатимуть нічого, прийматимуть змінну відповідного типу.

Одним з способів опису та представлення об’єктно-орієнтованих програм взагалі та класів зокрема, є використання UML-нотації (див. дод. 1). Наведемо представлення розроблюваного класу в UML:




*Класи в C++*. Класи об’являються з використанням ключового слова **class**, що визначає появу нового типу, який поєднуватиме дані та методи між собою. Клас являє логічну абстракцію і задає структуру деякої предметної області, а об’єкт — це конкретний екземпляр, що реалізує таку структуру.

Об’явлення класу в C++ схоже на об’явлення структури:

```
class ім’я_класу {
        приватні атрибути та методи;
специфікатор доступу: //public, private або protected
        атрибути та методи;
специфікатор доступу:
        атрибути та методи;
...
    } список_об’єктів_класу;
```

Специфікатори доступу можуть чередуватись в довільному порядку, область дії одного специфікатору простягається до наступного або до кінця об’явлення класу.

Для зручності подальшої підтримки та сприйняття об’єктно-орієнтованого коду програми слід розділяти об’явлення класу та його реалізацію, для цього об’явлення класу слід надавати в заголовкових файлах (.h, .hpp), а реалізацію — в файлах вихідного коду (.cpp).

Наведемо приклад коду об’явлення та реалізації класу “Робітник”:

- заголовковий файл «Employee.h»

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <cstring>

class Employee
{
   char name[25];
   int age;
   char position[25];
   float salary;
   int stage;
   static const int maxlen = 255;
   int date[maxlen][3];
   std::string jobList[maxlen];
   int job_index = 0;
   void showDate(int index);
   void calculateStage(int currentYear, int currentMonth, int currentDay);

   public:
      void setName(char *n);
      void getName(char *n);
      void setAge(int s);
      int getAge();
      void setPosition(char *p, int year, int month, int day);
      void getPosition(char *p);
      void setSalary(float s);
      float getSalary();
      void showCareer();
```

```
        int getStage(int currentYear, int currentMonth, int currentDay);
};

#endif // EMPLOYEE_H
```

- файл з реалізацією класу «Employee.cpp»

```
#include "Employee.h"

void Employee::setName(char* n)
{
    strcpy(name, n);
}

void Employee::getName(char* n)
{
    strcpy(n, name);
}


void Employee::setAge(int s)
{
    age = s;
}

int Employee::getAge()
{
    return age;
}

void Employee::setPosition(char* p, int year, int month, int day)
{
    strcpy(position, p);
    date[job_index][0] = year;
    date[job_index][1] = month;
    date[job_index][2] = day;
    jobList[job_index] = position;
    calculateStage(year, month, day);
    job_index++;
}

void Employee::getPosition(char* p)
{
    strcpy(p, position);
}

float Employee::getSalary()
{
    return salary;
}
```

```
void Employee::setSalary(float s)
{
    salary = s;
}

void Employee::showCareer()
{
    for(int i=0; i<job_index; i++)
    {
        std::cout << "Job " << i+1 << ": " << jobList[i] << ", started from: ";
        showDate(i);
    }
}

void Employee::showDate(int index)
{
    std::cout << date[index][0] << '.' << date[index][1] << '.' << date[index][2] << std::endl;
}

void Employee::calculateStage(int currentYear, int currentMonth, int currentDay)
{
    if(job_index == 0)
        stage = 0;
    else
    {
        int daysDelta = currentDay - date[0][2];
        if(daysDelta < 0)
            currentMonth -= 1;
        int monthsDelta = currentMonth - date[0][1];
        if(monthsDelta < 0)
            currentYear -= 1;
        stage = currentYear - date[0][0];
    }
}

int Employee::getStage(int currentYear, int currentMonth, int currentDay)
{
    if((currentYear != date[job_index][0]) || (currentMonth != date[job_index][1]) ||
(currentDay != date[job_index][2]))
        calculateStage(currentYear, currentMonth, currentDay);
    return stage;
}
```

- та файл «main.cpp» з прикладом створення і використання класу

```
#include <iostream>
#include <Employee.h>

using namespace std;
```

```
int main()
{
    Employee ivan;
    char name[25] = {"Ivan Ivanov"};
    ivan.setName(name);
    ivan.setSalary(10000);
    ivan.setPosition((char *)"Manager", 2004, 3, 1);
    char new_name[25];
    ivan.getName(new_name);
    cout << new_name << endl;
    cout << ivan.getSalary() << endl;
    ivan.setPosition((char *)"Director", 2010, 8, 21);
    ivan.showCareer();
    cout << ivan.getStage(2020, 2, 18) << endl;
    return 0;
}
```

**Лабораторне заняття 3.**

**Хід виконання завдання:**

1. Змоделювати клас для заданого поняття.
2. Описати клас в UML-нотації.
3. Реалізувати клас в вигляді програмного коду.
4. Написати програму, в якій користувач матиме можливість проводити маніпуляції зі створеним класом.

**Варіанти завдань.**

1. Вектор в просторі.
2. Прямокутник.
3. Раціональне число.
4. Дата.
5. Конус.
6. Пряма.
7. Мішане число.
8. Відрізок.
9. Комплексне число.
10. Ламана.
11. Вектор на площині.
12. Трапеція.
13. Матриця.
14. Трикутник.
15. Час.
16. Циліндр.
17. Багатокутник.
18. Сфера.

**Додаткові завдання.**

Для класу “Робітник”:

- замінити представлення кар’єрної історії з масиву на клас;
- представлення дати в вигляді класу з методами порівняння дат і виведення дати в консоль в вигляді строкової змінної.

**Додаток 1**. Основи UML

*UML* (*unified modeling language*) — узагальнена мова опису предметних областей, зокрема, програмних систем, об’єктно-орієнтованого аналізу і проєктування. Використовується для візуалізації, специфікації, конструювання і документування програмних систем [1]. В контексті об’єктно-орієнтованого програмування та аналізу в UML містяться такі базові компоненти:

1. Сутності, які являють абстракції, що складають модель.

2. Зв’язки, які пов’язують сутності між собою.

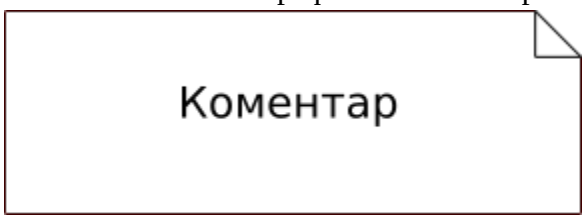
3. Діаграми для статичного представлення структури, згруповують набори сутностей в поняття. Являють зв’язний граф вершин (сутностей) і їх відносин (зв’язків). UML включає 13 базових діаграм, однією з найважливіших є діаграма класів, яка відображає набір класів, інтерфейсів та їх зв’язки.

Види сутностей:

*Структурні* — концептуальні або фізичні елементи системи, основним видом в структурних сутностях є *клас*.

Поведінкові — динамічні елементи діаграм, основною є *взаємодія*, яка позначає обмін повідомленнями між об’єктами діаграми, позначається в вигляді прямої зі стрілкою, над якою вказується ім’я операції:

*Анотуючі* — елементи, що пояснюють частини UML-моделей, головним видом є *примітка*, що позначається в вигляді прямокутника з загнутим правим верхнім кутом, всередині якого розміщується текстовий або графічний коментар:




Структурна сутність *клас* — опис набору об’єктів з однаковими властивостями (атрибутами), операціями (методами), зв’язками та поведінкою. Графічно зображається в вигляді прямокутника, що розділений на 3 блока горизонтальними лініями, порядок блоків (зверху-униз):

1. Ім’я класу.

2. Атрибути (властивості) класу.

3. Операції (методи) класу.

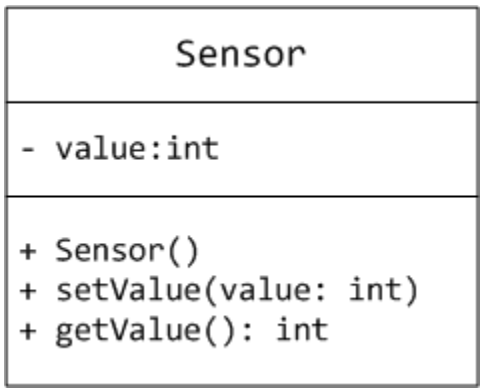
Додатково, для атрибутів та операцій може вказуватись один з трьох типів видимості:

- **public** позначається знаком “+” (плюс) перед назвою атрибуту або методу;
- **protected** позначається знаком “#” (решітка) перед назвою атрибуту або методу;
- **private** позначається знаком “-” (мінус) перед назвою атрибуту або методу.

Кожен клас повинен мати унікальну назву, що відрізнятиме його від інших класів. Ім’я класу — текстова строка, яка може складатись з будь-якого числа букв, цифр та інших знаків за винятком двокрапки та крапки і може бути складатись з декількох строк. Кращою практикою є короткі (за можливості) імена класів, які складаються з назв понять системи, що моделюється. Одним з підходів до іменування класів є так званий *CamelCase* (верблюже письмо), згідно якому кожне слово в назві класу пишеться з великої літери, наприклад: Sensor, TemperatureSensor або VelocityVectorField тощо.

Назви абстрактних класів записуються курсивом.

Приклад зображення класу в UML:

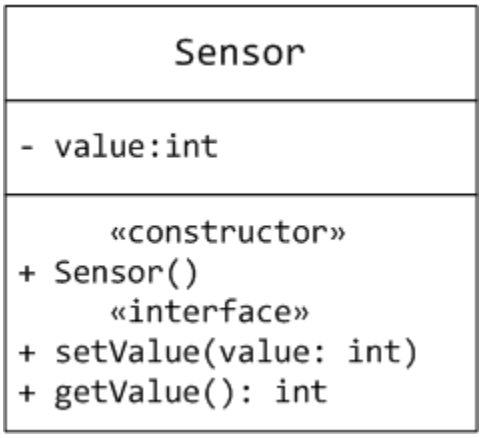



*Атрибут* — іменована властивість класу, що задає діапазон можливих значень для екземпляра атрибута, цю властивість матимуть усі екземпляри заданого класу. Кожен клас може містити будь-яку кількість атрибутів або не містити жодного. В останньому випадку блок атрибутів залишається порожнім. Імена атрибутів також складаються з назв понять предметної області, але записуються в варіації *lowerCamelCase* (нижнє верблюже письмо), за якою усі слова в назві атрибуту окрім першого починаються з великої літери, наприклад: value, rangeMaximum або pointPositionX.

Для атрибутів можна вказувати тип, кількість елементів (якщо атрибут є масивом), початкове значення.

*Метод* — іменована реалізація деякої функціональної операції класу. Клас може мати будь-яку кількість методів, або не мати жодного, в такому разі блок операцій залишається порожнім. Для іменування методів також може використовуватись *lowerCamelCase* але в якості першого слова найчастіше використовується дієслово, наприклад: getValue, isEqual, saveFileAndExit. Для методу можливо вказувати його сигнатуру, що включає імена, типи та значення за замовченням всіх параметрів та тип значення, що повертається.

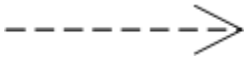
В процесі зображення класу необов’язково відображати усі його атрибути та методи, для конкретного представлення обираються найважливіші з них, але відповідний список атрибутів або методів в такому разі повинен закінчуватись трьома крапками. Для наочності довгі списки атрибутів або методів можливо згруповувати з використанням *стереотипів* — назв груп атрибутів або методів, які за ними слідують. Стереотипи вказуються в кутових дужках:




Відносини між класами в UML.

Існує 4 типи зв’язків:

1. Залежність — зв’язок між двома елементами моделі, в якій зміна одного елемента (незалежного) призводить до зміни поведінки другого елемента (залежного). Являє пунктирну лінію, іноді зі стрілкою, що направлена до тієї сутності, що є незалежною:



2. Асоціація — відображає як об’єкти однієї сутності (класу) пов’язані з об’єктами іншої сутності. Зображається в вигляді суцільної лінії зі стрілкою, яка вказує напрямок асоціації. Зазвичай, асоціації містять ім’я, яке розміщається над стрілкою.

Асоціації можуть бути множинними, в такому випадку на обох кінцях стрілки позначаються діапазони цілих чисел, які вказують на можливу кількість пов’язаних об’єктів, наприклад: точна кількість 3, нуль або один 0..1, будь-яка кількість 0..* або *, один або декілька 1..*, або діапазон цілих 2..5.

Агрегація (за посиланням) — особливий різновид асоціацій, що відображає структурний зв’язок цілого з його складовими. Використовується, коли один клас є контейнером для інших класів. Відношення агрегації не може включати більше двох класів (контейнер та його вміст). Знищення екземпляру класу ціле не призводить до знищення екземплярів класів, які він містить. Позначається пустим ромбом на класі “ціле” та прямою, яка йде від ромбу до класу “частина”:

Композиція, або агрегація за значенням — більш строгий варіант агрегації, в якому знищення екземпляру класу-контейнеру призводить до знищення екземплярів класів, які в ньому містяться. Графічне позначення:

3. Узагальнення — використовується для відображення наслідування, в якому елемент-нащадок будується за специфікаціями батьківського елемента. Нащадок розділятиме структуру і поведінку батьківського об’єкту. Графічно позначається в вигляді лінії з пустою стрілкою, яка вказує на батьківський елемент:

4. Реалізація — зв’язок між двома класами, в якому один з них (постачальник) визначає угоду, якої слідує другий клас (клієнт). Таким типом зв’язків позначаються зв’язки між інтерфейсами та класами, які реалізують ці інтерфейси. Позначається пунктирною лінією від клієнта з пустою стрілкою, яка вказує на постачальника: