Лекція 15.
Наслідування.

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

План на сьогодні

1. Основні принципи ООП. Що таке наслідування?
2. Захищені члени (protected)
3. Специфікатори доступу при наслідуванні
4. Багаторівневе наслідування
5. Множинне наслідування
6. Ієрархічне наслідування
7. Діамантова проблема. Віртуальне наслідування

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Основні приципи ООП

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Інкапсуляція

Інкапсуляція - принцип ООП, який полягає в об’єднанні

- **Даних -** параметрів та змінних, які відображають внутрішній стан об’єкта
- **Методів -** функцій-членів класу, які визначають поведінку об’єкту

в одну сутність (клас).

Інкапсуляція дозволяє обмежити прямий доступ до внутрішніх даних, надаючи доступ через публічні методи.

```
class Point{
protected:
    double x;
    double y;
public:
    Point(double xCoord = 0, double yCoord = 0);
    double getX() const;
    double getY() const;
    void print() const;
    double distanceTo(const Point& other) const;
};
```

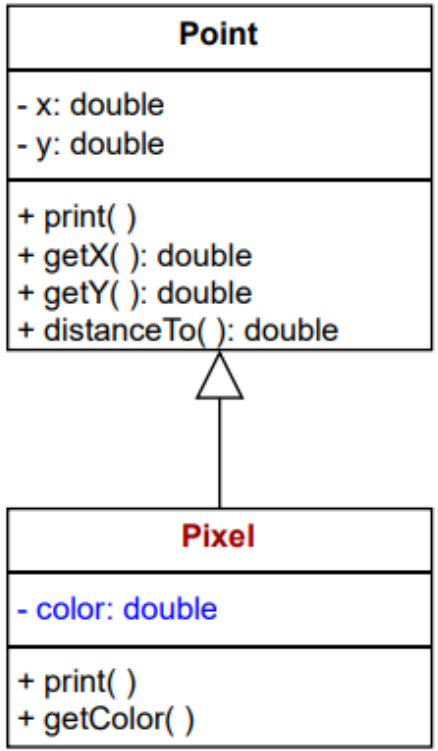
Наслідування

Наслідування - принцип ООП, який дозволяє одному класу (похідному або підкласу) отримувати властивості та методи іншого класу (базового або батьківського). Це допомагає:

- ❏ Уникати дублювання коду
- ❏ Створювати ієрархію класів
- ❏ Розширювати функціональність базового класу

Point: характеристика стану об'єкта – координати точки на площині; робота з точкою – через виклик методів

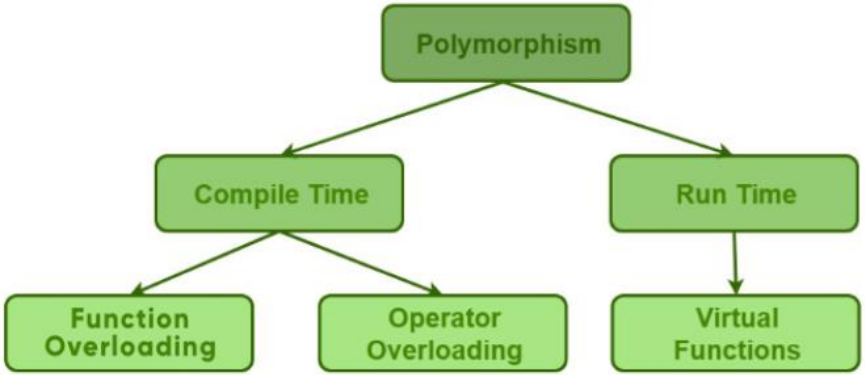
Pixel: дані об'єкта **Point** доповнюються даними про колір і методами, які працюють з кольором об'єкта



Поліморфізм

Слово **«поліморфізм»** означає **«багато форм»**. Простіше кажучи, поліморфізм можна визначити як здатність об’єкту відображатися в більш ніж одній формі.

Поліморфізм — це як універсальний ключ, який підходить до різних замків, або людина, яка в різних ситуаціях виконує різні ролі (батько, чоловік, співробітник).
Це механізм, що дозволяє об'єктам різних класів оброблятись через один і той самий інтерфейс, забезпечуючи різну поведінку залежно від конкретного типу об'єкта.
Приклад: Один метод **draw()** може малювати різні фігури (**Circle**, **Rectangle**, **Triangle**), залежно від того, до якого об'єкта він застосовується.



Що таке наслідування?

Що таке наслідування?

Наслідування (успадкування) є однією з ключових особливостей об'єктно-орієнтованого програмування в C++. Воно дозволяє створювати новий клас (називається **дочірнім**, **похідним** або **підкласом**) на основі існуючого класу (називається **батьківським, базовим** або **суперкласом**).

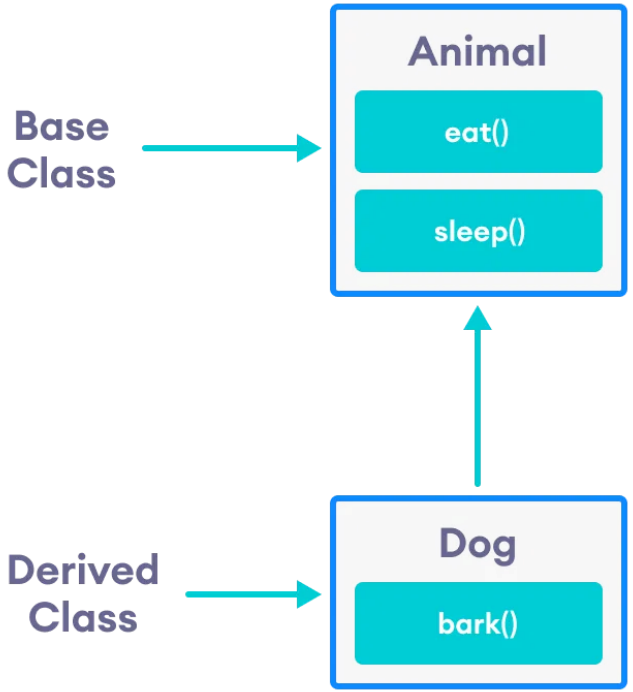
Похідний клас успадковує функціональність базового класу та може мати додаткові можливості або змінювати поведінку базового класу.

```
class назва_похідного_класу : специфікатор_доступу назва_базового_класу
{
    // тіло класу ...
};
```

Приклад наслідування

```
class Animal {
    public:
    // eat() function
    // sleep() function
};
class Dog : public Animal {
    public:
    // bark() function
};
```

Тут клас Dog успадковується від класу Animal. Оскільки Dog є похідним від Animal, члени Animal доступні в Dog.

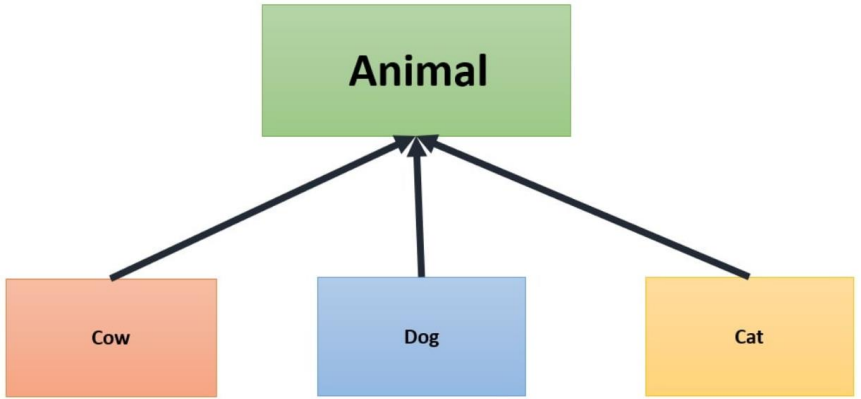


Відношення "є" (is-a)

Наслідування є **відношенням "є"**. Ми використовуємо наслідування лише тоді, коли між двома класами існує **відношення "є"**.

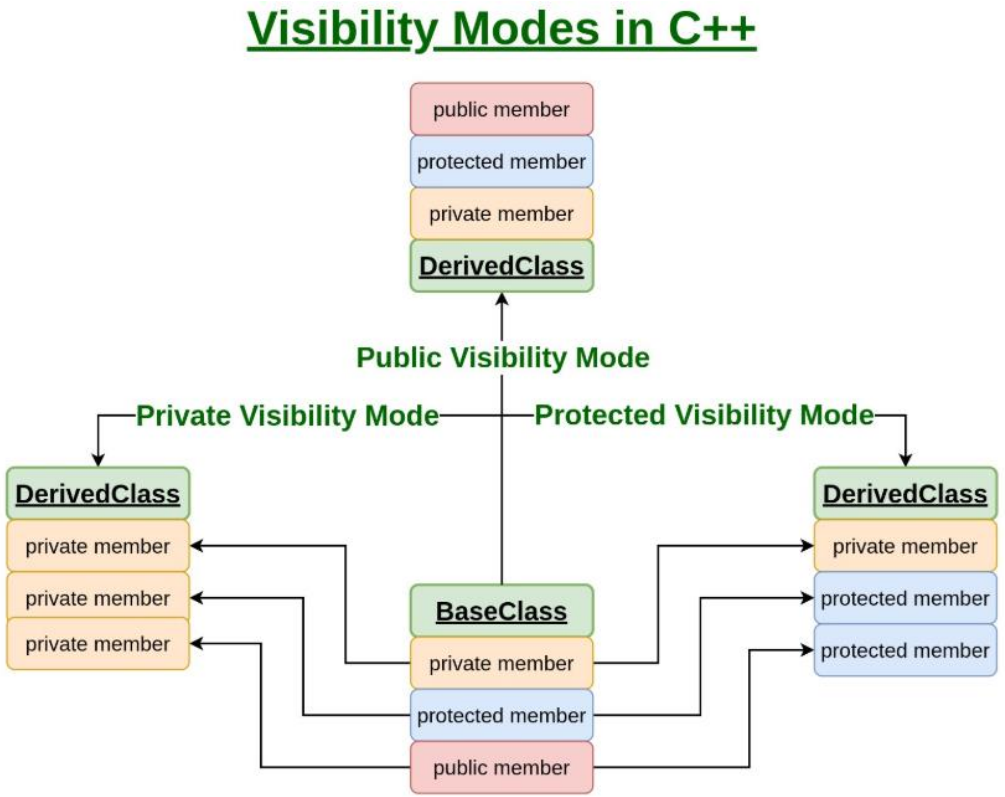
Ось кілька прикладів:

- Автомобіль є транспортним засобом.
- Апельсин є фруктом.
- Хірург є лікарем.
- Собака є твариною.
- Трикутник є фігурою.



Специфікатори доступу для наслідування

- Похідний клас не має доступу до private полів та методів базового класу!
- Специфікатор наслідування за замовчуванням private.



Специфікатор наслідування за замовчуванням

```cpp
class Base {
public:
    int publicVar = 10;
protected:
    int protectedVar = 20;
private:
    int privateVar = 30;
};

class Derived : Base { // Private наслідування
public:
    void show() {
        cout << "publicVar: " << publicVar << endl;    // ✅ Доступний у похідному класі
        cout << "protectedVar: " << protectedVar << endl; // ✅ Доступний у похідному класі
        // cout << privateVar; ❌ Помилка! private-члени базового класу НЕ доступні
    }
};

int main() {
    Derived obj;
    obj.show(); // ✅ Все працює

    // ❌ ПОМИЛКИ:
    // cout << obj.publicVar; // ❌ publicVar став private у Derived
    // cout << obj.protectedVar; // ❌ protectedVar теж став private

    return 0;
}
```

Особливості наслідування

- Всі public та protected члени (поля і методи) базового типу, стають автоматично членами похідного типу
- Не наслідуються: конструктори, деструктори, оператори присвоєння, дружні функції і класи, приватні члени не доступні у похідному класі
- У конструкторах похідного класу потрібно передбачити виклики конструкторів базових класів згідно списку наслідування зліва направо. Це здійснюється у списку ініціалізації, у іншому випадку викликається конструктор за замовчуванням базового класу
- виклик деструкторів базових типів здійснюється у порядку, зворотньому до виклику конструкторів
- функціональність базових типів у похідних типах може змінюватися за рахунок перевизначення методів і поліморфізму
- усунення неоднозначності виклику методів:
    - задання області видимості конкретного класу
    - приведення типу

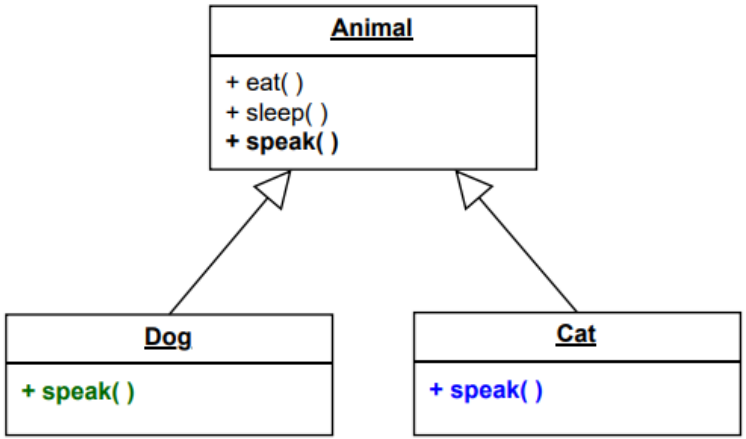
Приклад 1

Порядок побудови похідних об’єктів. Статичний поліморфізм.

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад 1 (Статичний поліморфізм)

- **`eat()`, `sleep()`, `speak()`**: Методи визначені в класі `Animal` та успадковані класами `Dog` і `Cat`, що дозволяє всім тваринам виконувати загальні дії (їсти, спати, говорити).
  ✅ **`speak()` в `Dog`**: Метод, який перевизначений (**overloaded**) і є специфічним для класу `Dog`, що дозволяє собакам гавкати.
  ✅ **`speak()` в `Cat`**: Метод, який перевизначений і є специфічним для класу `Cat`, що дозволяє котам мявкати.
- 📌 класи `Dog` та `Cat` можуть виконувати спільні дії, такі як їсти та спати, а також їхні унікальні поведінки (гавкати та мявкати).
- **Конструктор базового** класу завжди викликається перед конструктором похідного.
- **Деструктор базавого** класу викликається після деструктору похідного.



Захищені члени (protected)

Захищені члени (protected)

- Модифікатор доступу **protected** особливо важливий у контексті наслідування в C++.
- Подібно до **private**-членів, **protected**-члени недоступні поза межами класу. Однак вони можуть бути доступні у **похідних класах** та у **функціях/класах-друзях**.

- Ми використовуємо **protected**-члени, якщо хочемо приховати дані класу, але при цьому дозволити їх наслідування **похідними класами**.

Приклад 2

Захищені поля та методи класу

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад 2 (захищені поля та методи)

- Змінна color і методи startSpeaking() та endSpeaking() є private в Animal, тому вони не доступні у похідних класах напряму. Однак вони можуть бути доступні через публічні або protected методи.

- У базовому класі Animal змінна type та метод speakInfo() оголошені як protected. Це означає, що вони не доступні у main(), але доступні у похідних класах Dog і Cat.

- Перевантажений operator<< для Dog і Cat є friend, тому він має доступ до protected змінної type напряму. Але до private змінної color operator<< отримує доступ через getColor(), бо вона закрита для похідних класів.

Специфікатори доступу при наслідуванні

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Специфікатори доступу при наслідуванні

У C++ наслідування можна здійснювати в різних режимах доступу.
Наприклад: class Derived : **public** Base

Це означає, що ми створили похідний клас Derived на основі базового класу Base у **публічному режимі** (public).

Альтернативно, ми також можемо успадковувати класи у **захищеному** (protected) або **приватному** (private) режимі.

Ці три ключові слова (public, protected, private) називаються **специфікаторами доступу** у C++ наслідуванні.

Публічне, захищене та приватне наслідування

- **Публічне наслідування (`public`)**
  - `public` члени базового класу залишаються `public` у похідному класі.
  - `protected` члени базового класу залишаються `protected` у похідному класі.
- **Захищене наслідування (`protected`)**
  - `public` та `protected` члени базового класу стають `protected` у похідному класі.
- **Приватне наслідування (`private`)**
  - `public` та `protected` члени базового класу стають `private` у похідному класі.

Примітка: Приватні (`private`) члени базового класу недоступні в похідному класі.

Приклад 3.
Публічне наслідування

Приклад 3

```
Private = 1
Protected = 2
Public = 3
```

Тут ми успадкували PublicDerived від Base у **публічному режимі** (public).

У результаті в PublicDerived:

- prot успадковується як protected.
- pub і getPVT() успадковуються як public.
- pvt є недоступним, оскільки він private у Base.

Оскільки private та protected члени недоступні з main(), нам потрібно створити **публічні функції** getPVT() і getProt() для їх доступу.

Зверніть увагу, що функція getPVT() визначена всередині Base, а функція getProt() — всередині PublicDerived.

- Це тому, що pvt, який є private у Base, **недоступний** у PublicDerived.
- Однак prot доступний у PublicDerived завдяки **публічному наслідуванню**. Тому getProt() може отримати доступ до protected змінної всередині PublicDerived.

Доступність при публічному наслідуванні

Доступність	Приватні члени (`private`)	Захищені члени (`protected`)	Публічні члени (`public`)
У базовому класі	Так	Так	Так
У похідному класі	Ні	Так	Так

Приклад 4.
Захищене наслідування

Приклад 4

Тут ми успадкували `ProtectedDerived` від `Base` у **захищеному режимі** (`protected`).

У результаті в `ProtectedDerived`:

- `prot`, `pub` і `getPVT()` успадковуються як `protected`.
- `pvt` є недоступним, оскільки він `private` у `Base`.

Оскільки `protected`-члени не можуть бути безпосередньо доступні ззовні класу, ми **не можемо використовувати** `getPVT()` із `ProtectedDerived`.

Саме тому нам потрібно створити функцію `getPub()` у `ProtectedDerived`, щоб отримати доступ до змінної `pub`.

Область видимості полів та методів батьківського класу можна змінити у похідному розмістивши using Base::{field/method name} до потрібної області видимості похідного класу private/protected/public

```
Private cannot be accessed.
Protected = 2
Public = 3
```

Доступність при захищеному наслідуванні

Доступність	Приватні члени (`private`)	Захищені члени (`protected`)	Публічні члени (`public`)
У базовому класі	Так	Так	Так
У похідному класі	Ні	Так	Так (успадковуються як `protected`)

Приклад 5.
Приватне наслідування

Приклад 5

Тут ми успадкували `PrivateDerived` від `Base` у **приватному режимі** (`private`).

У результаті в `PrivateDerived`:

- `prot`, `pub` і `getPVT()` успадковуються як `private`.
- `pvt` недоступний, оскільки він є `private` у `Base`.

Оскільки `private`-члени не можуть бути безпосередньо доступними ззовні класу, ми **не можемо використовувати** `getPVT()` у `PrivateDerived`.

Саме тому потрібно створити функцію `getPub()` у `PrivateDerived`, щоб отримати доступ до змінної `pub`.

```
Private cannot be accessed.
Protected = 2
Public = 3
```

Доступність при приватному наслідуванні

Доступність	Приватні члени (`private`)	Захищені члени (`protected`)	Публічні члени (`public`)
У базовому класі	Так	Так	Так
У похідному класі	Ні	Так (успадковуються як `private`)	Так (успадковуються як `private`)

Множинне, багаторівневе, ієрархічне та віртуальне наслідування

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Множинне, багаторівневе, ієрархічне та віртуальне наслідування

Наслідування є однією з основних особливостей об'єктно-орієнтованої мови програмування. Воно дозволяє розробникам програмного забезпечення створювати новий клас на основі існуючого класу. Похідний клас успадковує властивості та функціональність базового класу (існуючого класу).

У C++ існують різні моделі наслідування.

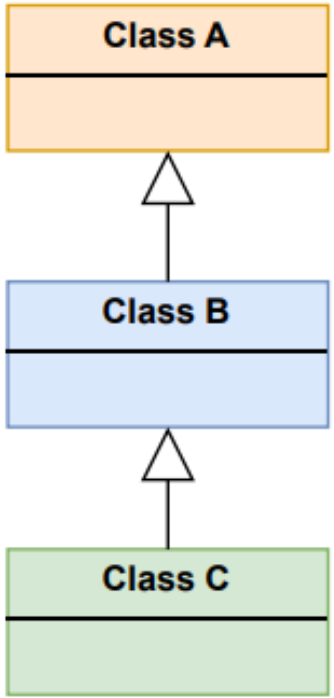
Багаторівневе наслідування

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Багаторівневе наслідування

У програмуванні на C++ ви можете не тільки створювати клас на основі базового класу, але й створювати клас на основі похідного класу. Ця форма наслідування називається **багаторівневим наслідуванням**.

Тут клас `B` успадковується від базового класу `A`, а клас `C` успадковується від похідного класу `B`.




```
class A {
//... .. ...
};
class B: public A {
//... .. ...
};
class C: public B {
//... ... ...
};
```

Приклад 6. Багаторівневе наслідування

Приклад 6

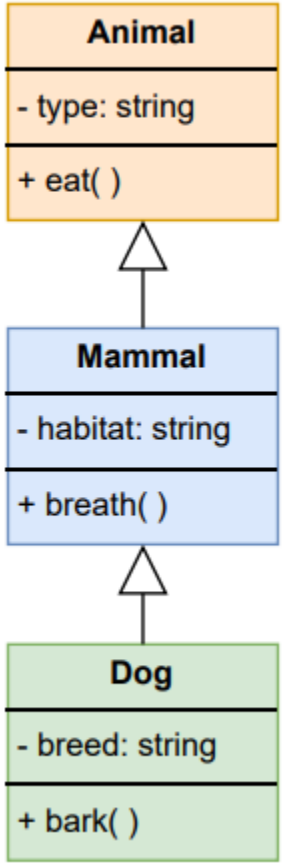
Порядок ініціалізації полів

1. **Спочатку викликаються конструктори базового класу (Animal)**
   - Ініціалізується type
2. **Потім конструктор проміжного класу (Mammal)**
   - Викликається конструктор Animal
   - Ініціалізується habitat
3. **Останнім викликається конструктор кінцевого класу (Dog)**
   - Викликається конструктор Mammal
   - Ініціалізується breed

Зворотний порядок виклику деструкторів:

1. Спочатку знищується об'єкт Dog
2. Потім Mammal
3. В кінці Animal

Поля protected доступні у похідних класах, private — тільки через гетери/сетері.



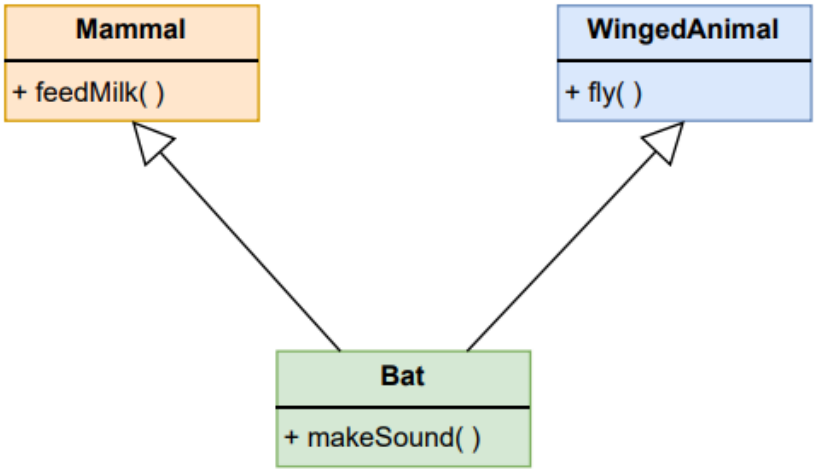
Множинне наслідування

FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY

Множинне наслідування

У програмуванні на C++ клас може успадковуватися від більше ніж одного батьківського класу.

Наприклад, клас `Bat` успадковується від базових класів `Mammal` (Ссавець) і `WingedAnimal` (Крилата тварина). Це логічно, оскільки кажан є одночасно ссавцем і крилатою твариною.



Приклад 7.
Множинне наслідування

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Двозначність у множинному наслідуванні

Найочевидніша проблема множинного наслідування виникає під час перевизначення функцій.

Припустимо, два базових класи мають одну й ту саму функцію, яка не перевизначена у похідному класі.

Якщо ви спробуєте викликати цю функцію за допомогою об'єкта похідного класу, компілятор видасть помилку. Це тому, що компілятор не знає, яку саме функцію викликати.

Цю проблему можна вирішити за допомогою **оператора розширення області видимості** (scope resolution operator), щоб вказати, яку функцію викликати — із `base1` чи `base2`.

```
class base1 {
  public:
     void someFunction( ) {....}
};
class base2 {
    void someFunction( ) {....}
};
class derived : public base1, public base2 {};
int main() {
    derived obj;
//    obj.someFunction() // Error!
    obj.base1::someFunction( );  // function of base1 class is called
    obj.base2::someFunction();   // function of base2 class is called.
}
```

Ієрархічне наслідування

Ієрархічне наслідування

Якщо більше ніж один клас успадковується від базового класу, це називається **ієрархічним наслідуванням**.

У ієрархічному наслідуванні всі загальні функції та властивості, які є спільними для дочірніх класів, включаються в базовий клас.

Наприклад:

- Класи `Physics`, `Chemistry`, `Biology` успадковуються від класу `Science`.
- Аналогічно, класи `Dog`, `Cat`, `Bird` успадковуються від класу `Animal`.

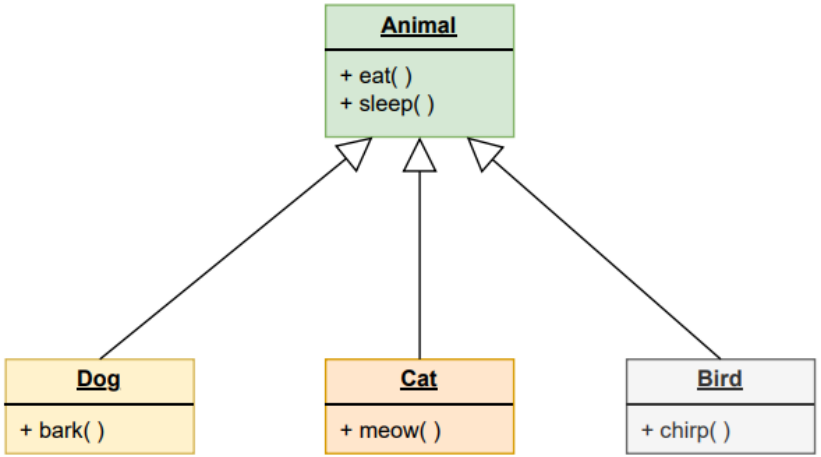
```
class base_class {
    ... .. ...
}
class first_derived_class: public base_class {
    ... .. ...
}
class second_derived_class: public base_class {
    ... .. ...
}
class third_derived_class: public base_class {
    ... .. ...
}
```

Приклад 8.
Ієрархічне наслідування

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад 8

- **Ієрархічне наслідування** дозволяє різним класам успадковувати спільну поведінку від одного базового класу.
- Це зменшує дублювання коду та покращує його **організацію та масштабованість**.
- Таку структуру часто використовують, коли кілька об'єктів мають **спільні атрибути** (наприклад, **усі тварини їдять і сплять, але видають різні звуки**).
- **Три похідні класи** (`Dog`, `Cat`, `Bird`) успадковують `Animal`, тому вони мають доступ до `eat()` і `sleep()`, а також додають власну поведінку (`bark()`, `meow()`, `chirp()`).



Даімантова проблема та Віртуальне наслідування

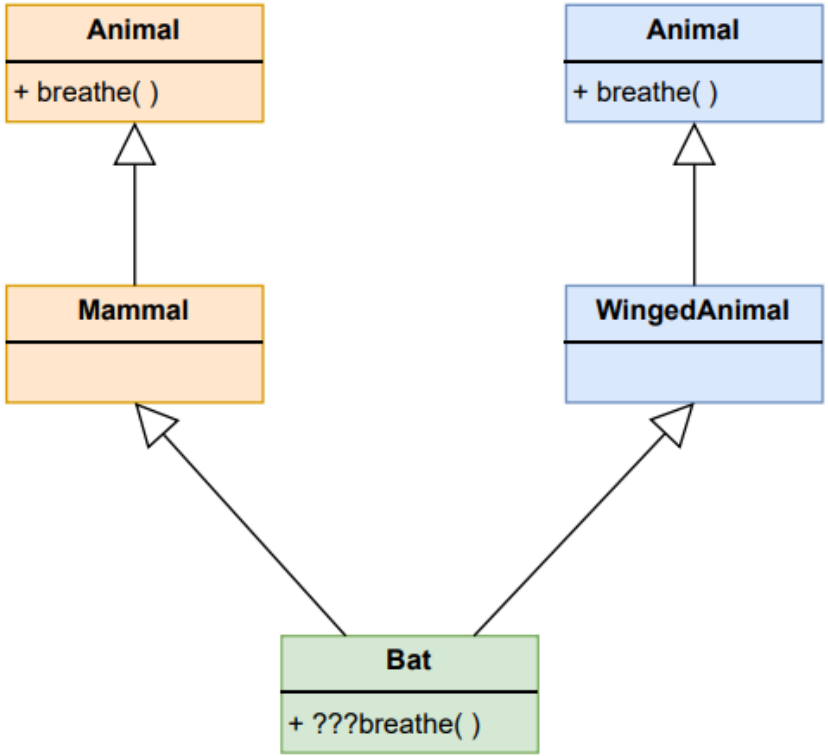
FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Діамантова проблема

Розглянемо ієрархію, де **Mammal** та **WingedAnimal** наслідують **Animal**, а **Bat** наслідує обидва.

Проблема:
Оскільки **Bat** успадковує **Mammal** і **WingedAnimal**, а вони мають базовий клас **Animal**, в результаті у **`Bat`** буде **дві копії** **`Animal`**.
Це спричиняє плутанину при виклику членів **`Animal`** через **`Bat`**.

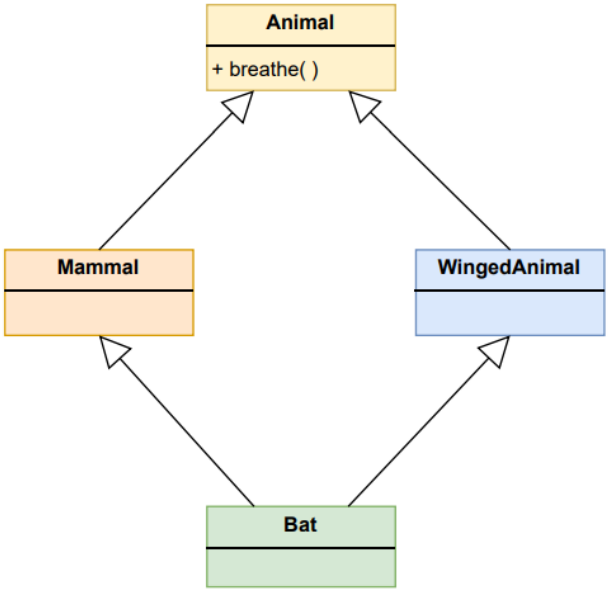
```
Creating Bat object...
Animal constructor called
Mammal constructor called
Animal constructor called
WingedAnimal constructor called
Bat constructor called

End of main, destructors will be called automatically.
Bat destructor called
WingedAnimal destructor called
Animal destructor called
Mammal destructor called
Animal destructor called
```



Віртуальне наслідування

Віртуальне наслідування — це техніка в C++, яка забезпечує, що похідні класи третього рівня (онуки) успадковують **лише одну копію** змінних-членів базового класу.

Розглянемо наступну ієрархію класів.



Віртуальне наслідування

Тут, коли клас `Bat` успадковується від кількох класів (`Mammal` і `WingedAnimal`), які мають спільний базовий клас `Animal`, він може успадкувати **кілька копій** базового класу. Це називається **проблемою діаманта**.

Ми можемо уникнути цієї проблеми, використовуючи **віртуальне наслідування**.

У цьому випадку `Derived1` і `Derived2` успадковують `Base` **віртуально**, що гарантує, що `Derived3` матиме **лише одну копію** змінних-членів `Base`, навіть якщо він успадковується від `Derived1` і `Derived2`.

```
class Derived1 : virtual public Base {
... .. ...
};
class Derived2 : virtual public Base {
... .. ...
};
class Derived3 : public Derived1, public Derived2 {
... ... ...
};
```

Приклад 9.
Віртуальне наслідування.

FACULTY OF APPLIED MATHEMATICS AND INFORMATICS
LVIV UNIVERSITY

Приклад 9

У цьому прикладі клас `Bat` успадковується від `WingedAnimal` і `Mammal`, які, у свою чергу, успадковуються від `Animal` з використанням **віртуального наслідування**.

Це гарантує, що під час створення екземпляра `Bat` **конструктор базового класу `Animal` викликається лише один раз**, і змінна `species_name` встановлюється у значення `"Bat"`.

Примітка: У наведеній програмі, якщо б **ключове слово `virtual` не використовувалося**, клас `Bat` успадкував би **декілька копій** змінних-членів класу `Animal`, що спричинило б **помилку**.

```
Animal constructor called
WingedAnimal constructor called
Mammals constructor called
Bat constructor called

It's a unique animal! Here are some details:
This animal belongs to the species: Bat
```

Дякую!

FACULTY OF APPLIED
MATHEMATICS AND
INFORMATICS
LVIV UNIVERSITY