

Лаб. 4. Успадкування та поліморфізм.

Мета: Навчитись використовувати функціонал мови C++ для створення складних гіллястих ієрархій класів.

Завдання: За аналогією до поданого прикладу розробити свою ієрархію класів, яка реалізує принципи успадкування та поліморфізму (мінімум 5 класів). Використати перевизначення та перевантаження методів. Тематику обрати самостійно.

1. Приклад діаграми класів

Розробка будь-якої ієрархії класів починається із її структури. На рисунку представлена діаграма класів для пропонованого прикладу.

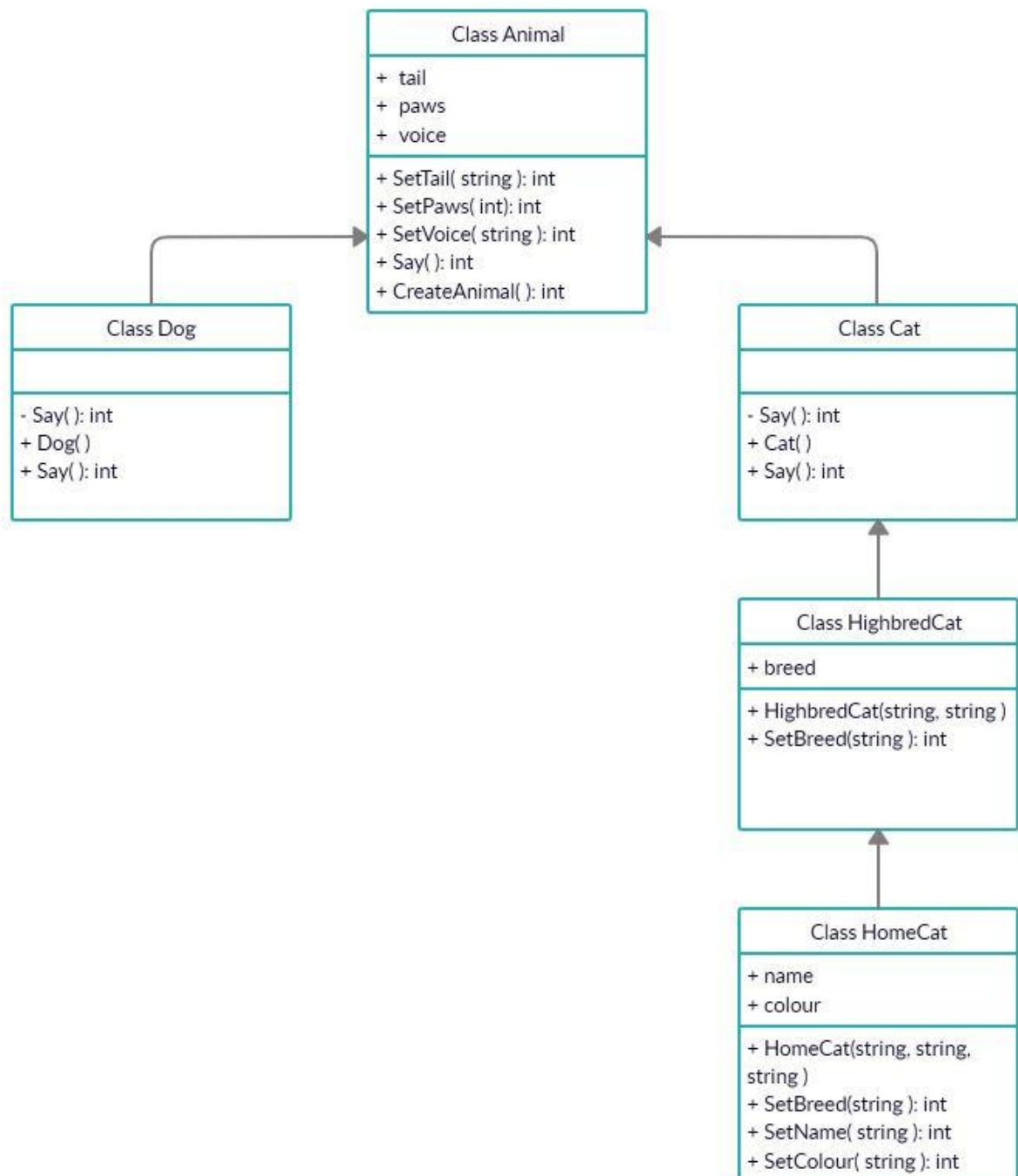


Рис. 1. Діаграма класів.

У нас є основний клас `Animal`, від якого наслідуються класи `Cat` та `Dog`. Наслідуються усі методи та поля, окрім методу `Say`, який перевизначений у кожному з класів.

Від класу `Cat` наслідується клас `HighbredCat`. У ньому є додаткове поле реалізований метод, який змінює це поле. Також у цьому класі реалізований свій конструктор.

Від класу `HighbredCat` наслідується клас `HomeCat` з реалізацією додаткових полів та методів. Варто зауважити, що у класу `HighbredCat` на діаграмі немає конструктору за замовчуванням.

2. Пояснення до коду у додатках

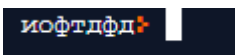
Реалізація нашого основного класу `Animal` представлена у стрічках 3-40. Для нашого випадку клас не є абстрактним, у ньому є 3 поля (`tail`, `paws` `voice`) із модифікатором доступу (по замовчуванню) `private`. Є методи-сетери для усіх цих параметрів (очевидно, вони мають бути доступні ззовні, тому мають модифікатор доступу `public`).

Також у класі `Animal` реалізовані методи `Say` (`public`) та `CreateAnimal` (`protected`).

У функції `main` створимо екземпляр нашого класу та викличемо метод `Say`:

```
int main() {  
    Animal *my_pet = new Animal();  
    int e = my_pet->Say();  
}
```

Ми отримаємо наступний вивід:



Якщо ми захочемо аналогічно викликати метод `CreateAnimal`, встановивши параметри які необхідно, отримаємо помилку:

```
int main() {  
    Animal *my_pet = new Animal();  
    int e = my_pet->SetTail("хвостатий хвіст");  
    e = my_pet->SetVois("странным");  
    e = my_pet->CreateAnimal();  
    e = my_pet->Say();  
}
```

Щоб її уникнути треба змінити модифікатор доступу наступним чином:

```
public:  
int CreateAnimal()
```

Тоді отримаємо наступний вивід:

```
Має 0 лапи та хвостатий хвіст.  
Говорить странным голосом: иофтдфд
```

Варто наголосити, що поле `paws` при ініціалізації класу заповнилось стандартним значенням (для `int` то число 0).

Як вже зазначалося, від класу `Animal` наслідуються два класи `Cat` (стрічки 42-59) та `Dog` (61-78). У конструкторі за замовчуванням кожного із цих класів встановлюються стандартні значення полів батьківського класу через його ж методи. Для класу `Cat` це стрічки 47-50 та викликаний метод `Say`. Цей метод у кожному із класів має свою реалізацію і не звертається до аналогічного методу батьківського класу. При створенні екземпляру класу `Cat`

```
int main() {  
    Cat *my_pet = new Cat();  
}
```

ми отримаємо наступний вивід:

```
Має 4 лапи та хвіст.  
Говорить няшним голосом: М-р-р-р... Мяу-мяу
```

Аналогічна ситуація для класу `Dog`:

```
int main() {  
    Dog *my_pet = new Dog();  
}
```

```
Має 4 лапи та обкусаний хвіст.  
Говорить грубим голосом: Гав... Гррр-р-гав
```

Ми використали просте наслідування, тому доступу до методів батьківського класу ззовні ми не маємо (усі методи отримали модифікатор `private`):

```
int main() {  
    Cat *my_pet = new Cat();  
    my_pet->SetPaws(5);  
}
```

Від класу `Cat` наслідуються наступний клас `HighbredCat`. У цьому класі є додаткове поле `breed` з модифікатором доступу `private` та функція для зміни цього параметру. Конструктор цього класу приймає стрічку як параметр та встановлює її у поле `breed`. Створивши екземпляр цього класу

```
int main() {  
    HighbredCat *my_pet = new HighbredCat("дворняга");  
}
```

ми отримаємо наступний вивід:

```
Має 4 лапи та хвіст.  
Говорить няшним голосом: М-р-р-р... М'яу-м'яу  
Порода кота дворняга: 
```

Можна бачити, що виконався і конструктор батьківського класу `Cat`, тому окрім породи виводиться уся інформація про котика)

Від класу `HighbredCat` наслідується наступний клас `HomeCat` (100-125). Він знов має поля та сетери для них. Але конструктор класу `HomeCat` не може викликати наш перевантажений конструктор, який приймає значення породи котика. Тому у класі `HighbredCat` створений пустий конструктор (97), який просто викличе конструктор вищого по ієрархії класу (у нашому випадку класу `Cat`).

Тому при створенні екземпляру нашого останнього класу таким чином

```
int main() {  
    HomeCat *my_pet = new HomeCat("дворняга", "Вася", "рижий");  
}
```

отримаємо наступний вивід:

```
Має 4 лапи та хвіст.  
Говорить няшним голосом: М-р-р-р... М'яу-м'яу  
То рижий кіт дворняга. Відкликається, коли кличеш Вася: 
```

4. Додаткове завдання

Винести опис класів та їх реалізацію у окремі файли (заголовковий файл та файл реалізації). У основному файлі залишити лише функцію `main`.

Додаток – лістинг коду

```
1  #include <iostream>  
2  
3  class Animal  
4  {  
5      std::string tail;  
6      int paws;  
7      std::string voice;  
8  
9      public:  
10     int SetTail (std::string tail_type)  
11     {  
12         tail = tail_type;  
13         return 1;  
14     };  
15  
16     int SetPaws (int number_of_paws)  
17     {
```

```

18 paws = number_of_paws;
19 return 1;
20 };
21
22 int SetVoice (std::string animal_voice)
23 {
24 voice = animal_voice;
25 return 1;
26 };
27
28 int Say ()
29 {
30 std::cout << "иофтдфд";
31 return 1;
32 };
33
34 protected:
35 int CreateAnimal()
36 {
37 std::cout << "Має " << paws << " лапи та " << tail << ". \n
    Говорить " << vois << " голосом: ";
38 return 1;
39 };
40 };
41
42 class Cat : Animal
43 {
44 public:
45 Cat()
46 {
47 int e = SetPaws(4);
48 e = SetTail("хвіст");
49 e = SetVois("няшним");
50 e = CreateAnimal();
51 e = Say();
52 };
53
54 int Say ()
55 {
56 std::cout << "М-р-р-р... Мяу-мяу";
57 return 1;
58 };
59 };
60
61 class Dog : Animal
62 {
63 public:
64 Dog()
65 {
66 int e = SetPaws(4);
67 e = SetTail("обкусаний хвіст");
68 e = SetVois("грубим");
69 e = CreateAnimal();

```

```

70 e = Say();
71 };
72
73 int Say ()
74 {
75     std::cout << "Гав... Гppp-p-гав";
76     return 1;
77 };
78 };
79
80 class HighbredCat : Cat
81 {
82     protected:
83     std::string breed;
84
85     public:
86     int SetBreed (std::string cats_breed)
87     {
88         breed = cats_breed;
89         return 1;
90     };
91
92     HighbredCat (std::string cats_breed)
93     {
94         int e = SetBreed (cats_breed);
95         std::cout << "\n Порода кота " << breed;
96     }
97     HighbredCat(){}
98 };
99
100 class HomeCat: HighbredCat
101 {
102     std::string name;
103     std::string colour;
104
105     public:
106     int SetName (std::string cats_name)
107     {
108         name = cats_name;
109         return 1;
110     };
111
112     int SetColour (std::string color)
113     {
114         colour = color;
115         return 1;
116     };
117
118     HomeCat (std::string cats_breed, std::string cats_name,
119             std::string color)
120     {
121         int e = SetBreed (cats_breed);
122         e = SetColour (color);

```

```
122 e = SetName (cats_name);
123 std::cout <<"\n To " << colour << " кіт " << breed << ".
    Відкликається, коли кличиш " << name;
124 }
125 };
126
127 int main()
128 {
129 HomeCat *my_pet = new HomeCat("дворяга", "Вася", "рижий");
130 }
```