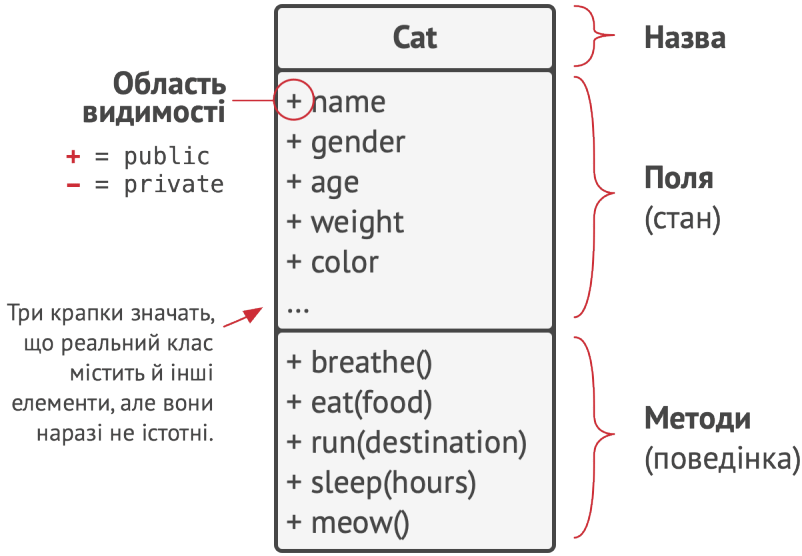
ВСТУП ДО ООП

Згадуємо ООП

***О**б'єктно-**о**рієнтоване **п**рограмування* — це методологія програмування, в якій усі важливі речі представлені **об'єктами**, кожен з яких є екземпляром того чи іншого **класу**, а класи утворюють **ієрархію** успадкування.

Об'єкти, класи

Ви любите кошенят? Сподіваюсь, що любите, тому я спробую пояснити усі ці речі на прикладах з котами.



Це UML-діаграма класу. У книзі буде багато таких діаграм.

Отже, у вас є кіт Пухнастик. Він є *об'єктом класу* `Кіт`. Усі коти мають однаковий набір властивостей: ім'я, стать, вік, вагу, колір, улюблену їжу та інше. Це — *поля* класу.

Крім того, всі коти поводяться схожим чином: бігають, дихають, сплять, їдять і муркочуть. Все це — *методи* класу. Узагальнено, поля і методи іноді називають *членами* класу.

> Значення полів певного об'єкта зазвичай називають його *станом*, а сукупність методів — *поведінкою*.

Pushystyk: Cat

```
name   = "Pushystyk"
sex    = "male"
age    =  3
weight =  5.5
color  =  gray
```

Murka: Cat

```
name   = "Murka"
sex    = "female"
age    =  1
weight =  3.5
color  =  white
```

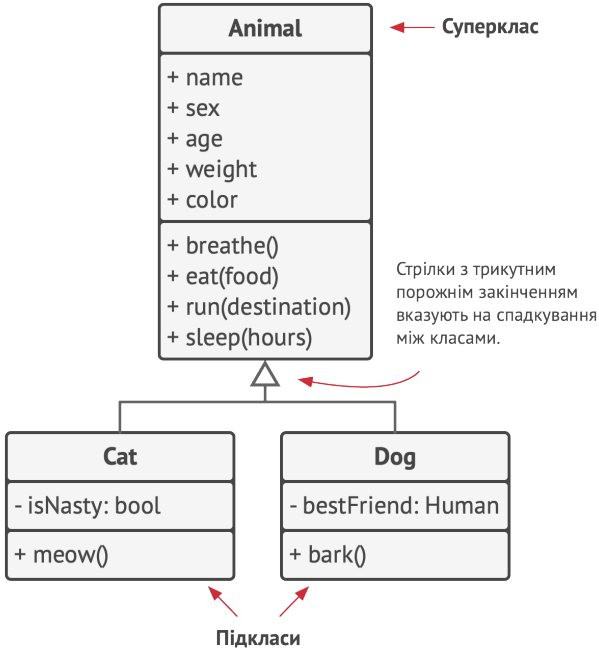
Об'єкти — це екземпляри класів.

Мурка, кішка вашої подруги, теж є екземпляром класу `Кіт`. Вона має такі самі властивості та поведінку, що й Пухнастик,

а відрізняється від нього лише значеннями цих властивостей — вона іншої статі, має інший колір, вагу тощо. Отже, **клас** — це своєрідне «креслення», на підставі якого будуються **об'єкти** — екземпляри цього класу.

Ієрархії класів

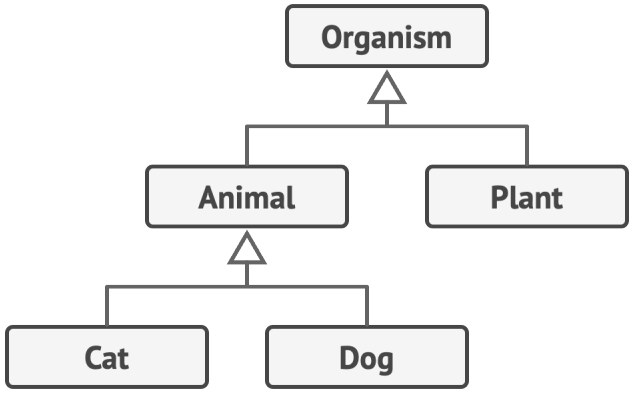
Ідемо далі. У вашого сусіда є собака Жучка. Як відомо, і собаки, і коти мають багато спільного: ім'я, стать, вік, колір є не тільки в котів, але й у собак. Крім того, бігати, дихати, спати та їсти можуть не тільки коти. Виходить так, що ці властивості та поведінка притаманні усьому класу `Тварини`.



UML-діаграма ієрархії класів. Усі класи на цій діаграмі є частиною ієрархії `Тварин`.

Такий батьківський клас прийнято називати **суперкласом**, а його нащадків — **підкласами**. Підкласи успадковують властивості й поведінку свого батька, тому в них міститься лише те, чого немає у суперкласі. Наприклад, тільки коти можуть муркотіти, а собаки — гавкати.

Ми можемо піти далі та виділити ще більш загальний клас живих `Організмів`, який буде батьківським і для `Тварин`, і для `Риб`. Таку «піраміду» класів зазвичай називають **ієрархією**. Клас `Котів` успадкує все, як з `Тварин`, так і з `Організмів`.



Класи на UML-діаграмі можна спрощувати, якщо важливіше показати зв'язки між ними.

Варто згадати, що підкласи можуть перевизначати поведінку методів, які їм дісталися від суперкласів. При цьому вони можуть, як повністю замінити поведінку методу, так і просто додати щось до результату виконання батьківського методу.

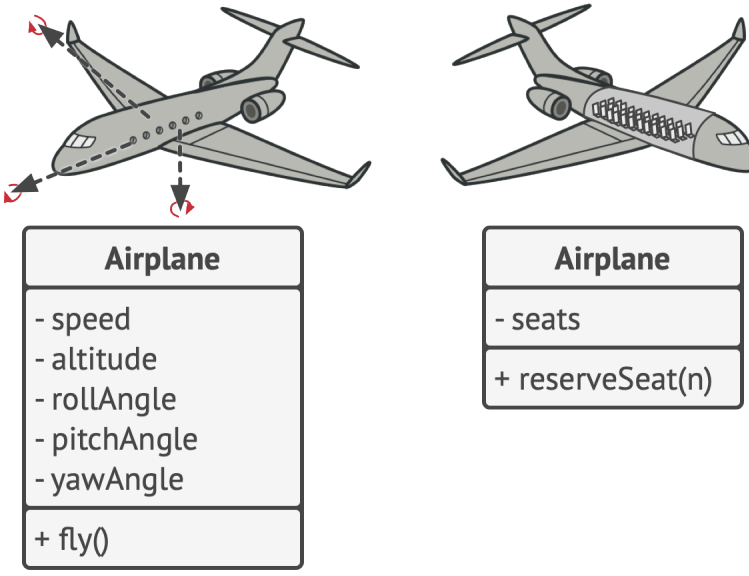
Наріжні камені ООП

ООП має чотири головні концепції, які відрізняють його від інших методологій програмування.

Абстракція

Коли ви пишете програму, використовуючи ООП, ви подаєте її частини через об'єкти реального світу. Але об'єкти у програмі не повторюють точно своїх реальних аналогів, та це й не завжди потрібно. Замість цього об'єкти програми всього лише *моделюють* властивості й поведінку реальних об'єктів, важливих у конкретному контексті, а інші — ігнорують.

Так, наприклад, клас `Літак` буде актуальним, як для програми-тренажера пілотів, так і для програми бронювання авіаквитків, але в першому випадку будуть важливими деталі пілотування літака, а в другому — лише розташування та наявність вільних місць усередині літака.



Різні моделі одного й того самого реального об'єкта.

Абстракція — це модель деякого об'єкта або явища реального світу, яка відкидає незначні деталі, що не грають істотної ролі в даному контексті.

Інкапсуляція

Коли ви заводите автомобіль, достатньо повернути ключ запалювання або натиснути відповідну кнопку. Вам не потрібно вручну з'єднувати дроти під капотом, повертати колінчастий вал та поршні, запускаючи такт двигуна. Всі ці деталі приховані під капотом автомобіля. Вам доступний лише простий інтерфейс: ключ запалювання, кермо та педалі. Таким чином, ми отримуємо визначення *інтерфейсу — публічної* (`public`) частини об'єкта, що доступна іншим об'єктам.

Інкапсуляція — це здатність об'єктів приховувати частину свого стану й поведінки від інших об'єктів, надаючи зовнішньому світові тільки визначений інтерфейс взаємодії з собою.

Наприклад, ви можете *інкапсулювати* щось всередині класу, зробивши його *приватним* (`private`) та приховавши доступ до цього поля чи методу для об'єктів інших класів. Трохи більш вільний, *захищений* (`protected`) режим видимості зробить це поле чи метод доступним у підкласах.

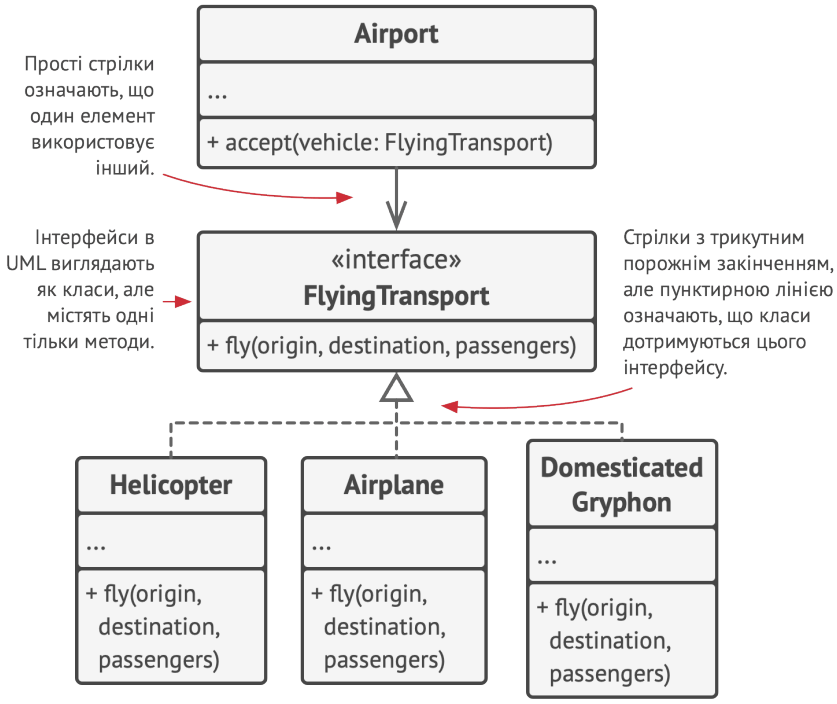
На ідеях абстракції та інкапсуляції побудовано механізми інтерфейсів і абстрактних класів/методів більшості об'єктних мов програмування.

> Багатьох вводить в оману те, що словом «інтерфейс» називають і публічну частину об'єкта, і конструкцію `interface` більшості мов програмування.

В об'єктних мовах програмування за допомогою механізму *інтерфейсів*, які зазвичай оголошують через ключове слово `interface`, можна явно описувати «контракти» взаємодії об'єктів.

Наприклад, ви створили інтерфейс `ЛітаючийТранспорт` з методом `летіти(звідки, куди, пасажири)`, а потім описали методи класу `Аеропорт` так, щоб вони приймали будь-які об'єкти з цим інтерфейсом. Тепер ви можете бути впевнені

в тому, що будь-який об'єкт, який реалізує інтерфейс чи то `Літак`, `Вертоліт` чи `ДресированийГрифон`, зможе працювати з `Аеропортом`.



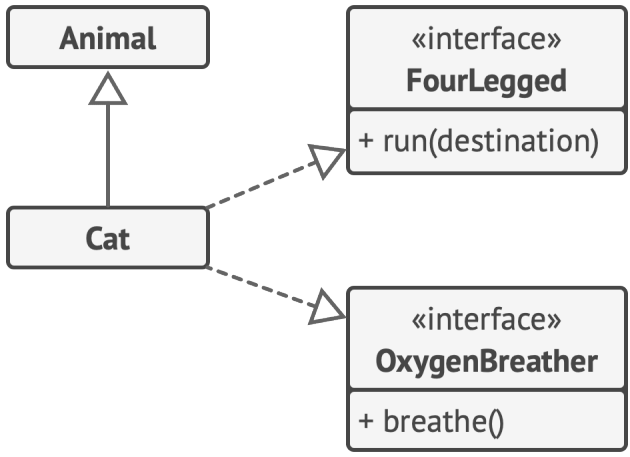
UML-діаграма реалізації та використання інтерфейсу.

Ви можете як завгодно змінювати код класів, що реалізують інтерфейс, не турбуючись про те, що `Аеропорт` втратить сумісність з ними.

Спадкування

Спадкування — це можливість створення нових класів на основі існуючих. Головна користь від спадкування — повто-

рне використання існуючого коду. Розплата за спадкування виражається в тому, що підкласи завжди дотримуються інтерфейсу батьківського класу. Ви не можете виключити з підкласу метод, оголошений його предком.

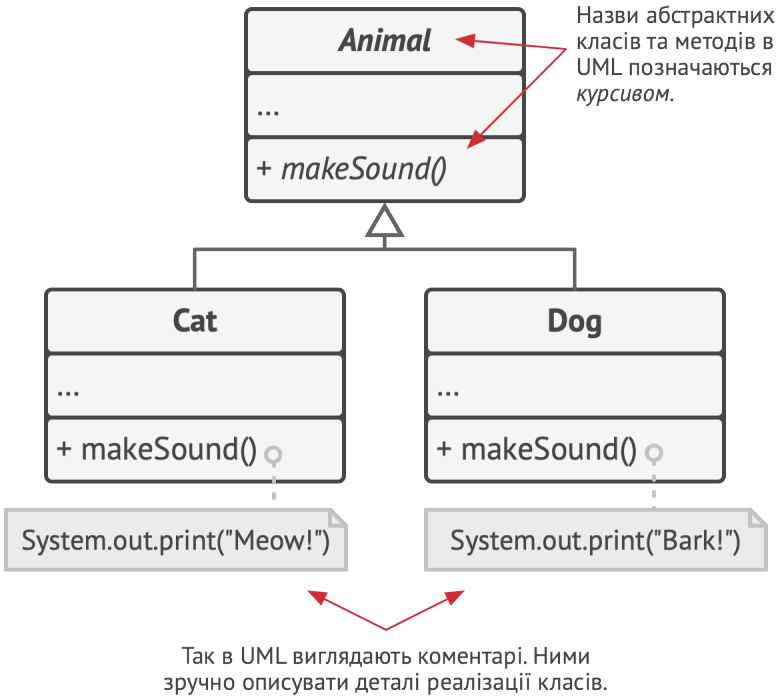


UML-діаграма одиничного спадкування проти реалізації безлічі інтерфейсів.

У більшості об'єктних мов програмування підклас може мати тільки одного «батька». Але, з іншого боку, клас може реалізовувати декілька інтерфейсів одночасно.

Поліморфізм

Повернемося до прикладів з тваринами. Практично всі Тварини вміють видавати звуки, тому ми можемо оголосити їхній спільний метод видавання звуків *абстрактним*. Усі підкласи повинні будуть перевизначити та реалізувати такий метод по-своєму.



Тепер уявіть, що ми спочатку помістили декількох собак і котів у здоровезний мішок, а потім із закритими очима будемо витягувати їх з мішка одне за одним. Витягнувши тваринку, ми не знаємо достеменно її класу. Але, якщо її

погладити, тваринка видасть звук залежно від її конкретного класу.

```
bag = [new Cat(), new Dog()];

foreach (Animal a : bag)
  a.makeSound()

// Meow!
// Bark!
```

Тут програмі невідомий конкретний клас об'єкта змінної `a`, але завдяки спеціальному механізмові, що називається *поліморфізм*, буде запущено той метод видавання звуків, який відповідає реальному класу об'єкта.

Поліморфізм — це здатність програми вибирати різні реалізації під час виклику операцій з однією і тією ж назвою.

Для кращого розуміння *поліморфізм* можна розглядати як здатність об'єктів «прикидатися» чимось іншим. У вищенаведеному прикладі собаки й коти прикидалися абстрактними тваринами.