9.1.1. Поняття класу та об'єкта в об’єктно-орієнтованому програмуванні; конструктор і деструктор, інтерфейс та реалізація

Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування (ООП, іноді об'єктно-зорієнто́ване програмува́ння; англ. Object-oriented programming, OOP) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єкто-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення[en]. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єкто-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний

отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Клас визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності (її атрибутами або властивостями) та діями, які вона здатна виконувати (її поведінкою, методами або можливостями). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єкто-орієнтовану програму. Зазвичай клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, своєю чергою, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами.

В об'єктно-орієнтованому програмуванні, **клас** — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції називають методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).

Класи та екземпляри класів.

Клас можна порівнювати з формою для випічки печива — форма одна, а печива можна випекти безліч. Печиво — це конкретні об'єкти, екземпляри класу печиво, яке може бути з різною начинкою. Поля дозволяють вмістити дані про певний реальний об'єкт, а методи здійснювати обробку цих даних. Наприклад, можна створити загальний клас Людина з полями Ім'я та Прізвище, рік народження, професія, зарплата. При створенні ж на основі класу конкретного екземпляру, дані поля заповнюються конкретними даними про певну людину. Обробкою цих даних можуть займатися відповідні методи. Наприклад, можна створити метод для обчислення віку людини, тощо.

На основі класів можна створювати підкласи, які успадковують властивості та поведінку батьківських класів. Таким чином можна створити цілу ієрархію класів. Різні мови дещо по різному реалізовують механізм успадкування. Існує множинне та одинарне успадкування. Множинне — це, коли підклас створюється на основі кількох безпосередніх батьків (як то в мові програмування C++). Одинарне успадкування — це коли клас може мати одного безпосереднього батька (мова програмування Java). Надкласи можуть мати свої надкласи, підкласи можуть також бути надкласами для певних класів.

Методи класів.

Через методи реалізується поведінка об'єктів. Практично вся робота з об'єктами відбувається через методи. Вони можуть змінювати стан об'єкта або ж просто надавати доступ до даних, які були інкапсульовані в об'єкті. Існує кілька видів методів, які мають деякі відмінності в різних мовах програмування. До методів та полів даних можна надавати різні права доступу, від яких залежатиме доступ до них з різних частин програмного коду. Права доступу та вид методів задаються модифікаторами при описі методів. Метод, який проводить створення та початкову ініціалізацію екземпляра класу називають конструктором класу. Метод, який проводить знищення об'єкта, називають деструктором класу.

Інтерфейс та реалізація, спадкування реалізації.

У програмуванні існує поняття програмного інтерфейсу, що означає перелік можливих обчислень, які може виконати та чи інша частина програми, включаючи опис того, які аргументи і в якому порядку потрібно передавати на вхід алгоритмам з цього переліку, а також що і в якому вигляді вони будуть повертати. Абстрактний тип даних інтерфейс придуманий для формалізованого опису такого переліку. Самі алгоритми, тобто дійсний програмний код, який буде виконувати всі ці обчислення, інтерфейсом не задається, програмується окремо та називається реалізацією інтерфейсу.

Програмні інтерфейси, а також класи, можуть розширюватися шляхом спадкування, яке є одним з важливих засобів повторного використання готового коду в ООП. Успадкований клас або інтерфейс буде містити в собі все, що зазначено для всіх його батьківських класів (в залежності від мови програмування та платформи, їх може бути від нуля до нескінченності). Наприклад, можна створити свій варіант текстового рядка шляхом успадкування класу «мій рядок тексту» від вже існуючого класу «рядок тексту», при цьому передбачається, що програмісту не доведеться заново переписувати алгоритми пошуку та інше, оскільки вони автоматично будуть успадковані від готового класу, і будь-який екземпляр класу «мій рядок тексту» може бути переданий не лише в готові методи батьківського класу «рядок тексту» для проведення потрібних обчислень, а й взагалі в будь-який алгоритм, здатний працювати з об'єктами типу «рядок тексту», оскільки екземпляри обох класів сумісні по програмним інтерфейсам.

Клас дозволяє задати не лише програмний інтерфейс до самого себе і до своїх екземплярів, але і в явному вигляді написати код, відповідальний за обчислення. Якщо при створенні свого нового типу даних успадковувати інтерфейс, то ми отримаємо можливість передавати примірник свого типу даних в будь-який алгоритм, який вміє працювати з цим інтерфейсом. Однак нам доведеться самим написати реалізацію інтерфейсу, тобто ті алгоритми, якими буде користуватися цікавий нам алгоритм для проведення обчислень з використанням нашого екземпляра. Водночас, при наслідуванні класу, автоматично успадковується готовий код під інтерфейс (це не завжди так, батьківський клас може вимагати реалізації якихось алгоритмів в дочірньому класі в обов'язковому порядку). В такій можливості успадковувати готовий код і проявляється те, що в об'єктно-орієнтованої програми тип даних клас визначає одночасно як інтерфейс, так і реалізацію для всіх своїх екземплярів.

Стан об'єкта, поняття областей доступу, конструктори.

Однією з проблем структурного програмування, з якою бореться ООП, є проблема підтримки правильного значення змінних програми. Часто різні змінні програми зберігають логічно пов'язані значення, і за підтримання цієї логічної зв'язності несе відповідальність програміст, тобто автоматично зв'язність не підтримується. Прикладом може слугувати пара прапорців «звільнений» та «очікує премії за підсумками року», коли за правилами відділу кадрів людина може бути водночас не звільненою і не очікувати премію, або не звільненою та очікувати премію, або звільненою і не очікувати премію, але не може бути одночасно звільненою і очікувати премію. Тобто будь-яка частина програми, яка проставляє прапорець «звільнений», завжди повинна знімати прапорець «чекає премії за підсумками року».

Хороший спосіб вирішити цю проблему — оголосити прапорець «звільнений» недоступним до зміни для всіх ділянок програми, крім одного спеціально обумовленого. У цій спеціально обумовленій ділянці все буде написано один раз і правильно, а всі інші

повинні будуть звертатися до цієї ділянки завжди, коли вони хочуть встановити або зняти прапорець «звільнений».

В об'єктно-орієнтованій програмі прапорець «звільнений» буде оголошено приватним членом деякого класу, а для його читання та зміни будуть написані відповідні публічні методи. Правила, що визначають можливість або неможливість безпосередньо змінювати будь-які змінні, називаються правилами завдання областей доступу. Слова «приватний» та «публічний» в цьому випадку є так званими «модифікаторами доступу». Вони називаються «модифікаторами» тому, що в деяких мовах вони використовуються для зміни раніше встановлених прав при спадкуванні класу. Спільно класи та модифікатори доступу задають область доступу, тобто у кожної ділянки коду, залежно від того, до якого класу вона належить, буде своя область доступу щодо тих чи інших елементів (членів) свого класу та інших класів, включаючи змінні, методи, функції, константи, тощо. Існує основне правило: ніщо в одному класі не може бачити приватних елементів іншого класу. Щодо інших, більш складних правил, у різних мовах існують інші модифікатори доступу та правила їх взаємодії з класами.

Майже кожному члену класа можна встановити модифікатор доступу (за винятком статичних конструкторів та деяких інших речей). У більшості об'єктно-орієнтованих мов програмування підтримуються такі модифікатори доступу:

- private (закритий, внутрішній член класу) — звернення до члену допускаються лише з методів того класу, у якому цей член визначений. Будь-які спадкоємці класу вже не зможуть отримати доступ до цього члену. Спадкування за типом private забороняє доступ з дочірнього класу до всіх членів батьківського класу, включаючи навіть public-члени (C++);
- protected (захищений, внутрішній член ієрархії класів) — звернення до члена допускаються з методів того класу, у якому цей член визначений, а також з будь-яких методів його класів-спадкоємців. Спадкування за типом protected робить всі public-члени батьківського класу protected-членами класу-спадкоємця (C++);
- public (відкритий член класу) — звернення до члена допускаються з будь-якого коду. Спадкування за типом public не міняє модифікаторів батьківського класу (C++);

Проблема підтримки правильного стану змінних актуальна і для вихідного моменту виставлення початкових значень. Для цього в класах передбачені спеціальні методи/функції, звані конструкторами. Жоден об'єкт (екземпляр класу) не може бути створений інакше, як шляхом виклику на виконання код конструктора, який поверне створений та правильно заповнений примірник класу. У багатьох мовах програмування тип даних «структура», як і клас, може містити змінні та методи, але екземпляри структур, залишаючись просто розміченими ділянками оперативної пам'яті, можуть створюватися в обхід конструкторам, що заборонено для примірників класів (за винятком спеціальних виняткових методів обходу всіх подібних правил ООП, передбачених в деяких мовах та платформах). У цьому виявляється відмінність класів від інших типів даних — виклик конструктора обов'язковий.

Практичний підхід.

У сучасних об'єктно-орієнтованих мовах програмування (в тому числі в php, Java, C++, Oberon, Python, Ruby, Smalltalk, Object Pascal) створення класу зводиться до написання деякої структури, що містить набір полів та методів (серед останніх особливу роль грають конструктори, деструктори, фіналізатори). Практично клас може розумітися

як якийсь шаблон, за яким створюються об'єкти — екземпляри цього класу. Усі примірники одного класу створені за одним шаблоном, тому мають один і той же набір полів та методів.

В об'єктно-орієнтованому програмуванні **протокол або інтерфейс** є звичайним засобом для незв'язаних об'єктів спілкуватися один з одним. Це визначення методів та цінностей, з яким об'єкти погоджуються для співпраці.

Наприклад, у Java, де протоколи називаються інтерфейсами, інтерфейс Comparable визначає метод compareTo(), який реалізовані класи повинні виконувати. Це означає, що окремий метод сортування, наприклад, може сортувати будь-який об'єкт, який реалізує інтерфейс Comparable, без необхідності знати будь-що про внутрішню природу класу (крім того, що два з цих об'єктів можна порівняти за допомогою compareTo()).

Протокол є описом:

- повідомлень, які розуміються об'єктом;
- аргументів, якими можуть надаватися ці повідомлення;
- типів результатів, до яких ці повідомлення повертаються;
- інваріантів, які зберігаються попри модифікації стану об'єкта;
- виняткових ситуацій, які вимагатимуть клієнтів обробляти об'єкт.

Якщо об'єкти повністю інкапсульовані, то протокол описує єдиний спосіб доступу до цих об'єктів іншими об'єктами.

Деякі мови програмування забезпечують явну мовну підтримку протоколів або інтерфейсів (Ada, C#, D, Dart, Delphi, Go, Java, Logtalk, Object Pascal, Objective-C, PHP, Racket, Seed7, Swift). У C++ інтерфейси відомі як абстрактні базові класи і реалізовані за допомогою чистих віртуальних функцій. Об'єктно-орієнтовані функції Perl також підтримують інтерфейси.

Попри те, що Go загалом не розглядається як об'єктно-орієнтована мова програмування, вона дозволяє визначати методи на типи, визначені користувачем. Go має типи "інтерфейсу", сумісні з будь-яким типом, який підтримує певний набір методів (для цього типу не потрібно явно вводити інтерфейс). Порожній інтерфейс interface{} сумісний з усіма типами.

Функціональне програмування та розподілені мови програмування також мають поняття протоколу, значення якого тонко відрізняється (тобто специфікація дозволеного обміну повідомленнями, акцент на обмін, а не на повідомлення). Ця різниця обумовлена дещо різними припущеннями функціонального програмування та парадигмами об'єктно-орієнтованого програмування. Зокрема, наступні також розглядаються як частина протоколу на таких мовах:

- допустимі послідовності повідомлень;
- обмеження, що покладаються на одного учасника зв'язку;
- очікувані ефекти, які відбудуться під час обробки повідомлення.

Класи типу в мовах, таких як Haskell, використовуються для багатьох речей, для яких використовуються і протоколи.

Інтерфейс і реалізація.

Складні задачі треба розділити на багато маленьких й передоручити їх дрібним субпідрядникам. Ніде ця ідея не проявляє себе так яскраво, як у проектуванні класів.

За своєю природою, клас - це генеральний контракт між абстракцією й всіма її клієнтами. Виразником зобов'язань класу служить його інтерфейс, причому в мовах із сильною типізацією потенційні порушення контракту можна виявити вже на стадії компіляції.

Ідея контрактного програмування приводить нас до розмежування зовнішнього вигляду, тобто інтерфейсу, і внутрішньої будови класу, реалізації. Головне в інтерфейсі - оголошення операцій, які підтримуються екземплярами класу. До нього можна додати оголошення інших класів, змінних, констант і виняткових ситуацій, що уточнюють абстракцію, яка виражається в класі. Навпаки, реалізація класу нікому, крім нього самого, не цікава. Реалізація складається у визначенні операцій, оголошених в інтерфейсі класу.

Ми можемо розділити інтерфейс класу на три частини:

- відкриту (public) - видиму всім клієнтам;
- захищену (protected) - видиму самому класу, його підкласам і друзям (friends);
- закриту (private) - видиму тільки самому класу і його друзям.

Різні мови програмування передбачають різні комбінації цих частин. Розробник може задати права доступу до тієї або іншої частини класу, визначивши тим самим зону видимості клієнта.

Зокрема, в C++ всі три перерахованих рівні доступу визначаються явно. На додаток до цього є ще й механізм друзів, за допомогою якого стороннім класам можна надати привілей бачити закриту й захищену область класу. Тим самим порушується інкапсуляція, тому, як і в житті, друзів треба вибирати обережно. В Ada оголошення можуть бути зроблені закритими або відкритими. В Smalltalk всі змінні - закриті, а методи – відкриті. В CLOS узагальнені функції відкриті, а слоти можуть бути закритими, хоча довідатися їх значення однаково можна.

Стан об'єкта задається в його класі через визначення констант або змінних, що поміщаються в його захищеній або закритій частині. Тим самим вони інкапсульовані, і їхньої зміни не впливають на клієнтів.

Уважний читач може запитати, чому ж подання об'єкта визначається в інтерфейсній частині класу, а не в його реалізації. Причини практичні: в іншому випадку необхідно було б об’єктно-орієнтовані процесори або дуже хитромудрі компілятори. Коли компілятор опрацьовує оголошення об'єкта, наприклад, таке:

DisplayItem item1;

він повинен знати, скільки відвести під нього пам'яті. Якби ця інформація зберігалася в реалізації класу, нам довелося б написати її повністю, перш, ніж ми змогли б задіяти його клієнтів. Тобто, весь смисл відділення інтерфейсу від реалізації був би загублений.

Константи й змінні, які складають подання класу, відомі під різними іменами. В Smalltalk їх називають змінні екземпляра, в Object Pascal - поля, в C++ - члени класу, а в CLOS - слоти. Ми часто будемо використовувати ці терміни як синоніми.

Окремий екземпляр класу (створюється після запуску програми та ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт

Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об'єкта називається станом. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкти (Сірко і Дружок) є екземплярами класу Собака.

Об'єкт в об'єктно-орієнтованому програмуванні

Об'єкт в об'єктно-орієнтованому програмуванні (ООП) — ключове поняття об'єктно-орієнтованих технологій проєктування та програмування; втілення абстрактної моделі окремої сутності (предмету або поняття), що має чітко виражене функціональне призначення в деякій області, належить до визначеного класу та характеризується своїми властивостями та поведінкою. Об'єкти є базовими елементами побудови програми — програма в ООП розглядається як сукупність об'єктів, що знаходяться у визначених відношеннях та обмінюються повідомленнями.

Поняття об'єкт в програмному забезпеченні вперше було введене в мові Simula в середині 1960-х років для моделювання реальності.

Властивості та поведінка об'єкта.

Властивості об'єкта визначаються його атрибутами (полями даних). Поточне значення атрибутів визначає поточний стан об'єкта у множині можливих станів.

Поведінка об'єкта визначається функціями (методами) об'єкта. Передача повідомлень між об'єктами (взаємодія об'єктів) призводить до виконання об'єктом, що отримав повідомлення, визначеної функції. Об'єкт також може надіслати повідомлення собі. В результаті отримання об'єктом повідомлення він змінює свій стан: на новий, якщо виконання операцій функції призвело до зміни значень атрибутів; або той самий, якщо атрибути не зазнали змін. В контексті отримання повідомлень та зміни станів об'єкт може розглядатись як автомат.

Загалом, поведінка та властивості об'єкта визначають його ідентичність, що вирізняє об'єкт серед інших об'єктів.

Характеристики об'єктів.

Об'єкти створюються у програмі під час її виконання і, зазвичай, існують у межах програми, але, в окремих випадках, можуть існувати й поза межами програми — наприклад, у вигляді файлів або пакетів даних. Механізми, що дозволяють існування об'єктів поза межами програми, підтримуються окремими ОО-мовами програмування.

Властивості об'єкта, зазвичай, доступні лише через його функції. При цьому вважається, що об'єкт є екземпляром класу як абстрактного типу даних. В окремих випадках, що, загалом, порушують принципи ООП, властивості об'єкта можуть бути загальнодоступними. Такі властивості, як правило, є константами.

Відповідно до властивостей об'єкта та його стану, функції поділяються на конструктори, селектори, модифікатори та деструктори:

- конструктори здійснюють первинну ініціалізацію об'єкта під час його створення;
- селектори повертають значення окремих властивостей;
- модифікатори змінюють значення окремих властивостей;

– деструктори скидають значення властивостей під час знищення об'єкта.

Представлення об'єктів.

Об'єкти, зазвичай, зберігаються в оперативній пам'яті під час виконання програми. При цьому вони представлені в пам'яті послідовністю значень атрибутів — структурою даних. Всі функції об'єктів зберігаються поза межами об'єктів і для функцій лише забезпечується контекст — можливість звернення до атрибутів вказаного об'єкта. В окремих ОО-мовах програмування дані об'єкта або об'єктів в пам'яті можуть бути перенесені поза межі пам'яті програми, наприклад — у зовнішній файл, і в подальшому — поновлені. При цьому об'єкт опиниться в тому ж стані, в якому він перебував на момент збереження.

Статичні (спільні для всіх об'єктів класу) атрибути об'єктів зберігаються поза межами структур даних об'єктів і не впливають на їх розмір у пам'яті.