Об'єктно-орієнтований підхід у розробці програмного забезпечення.

***Виконав*** : Беліч Олександр

***Група,факультет*** : К-21, ІТЕ

***Лектор*** : Артеменко О.І.

## Зміст:

1. Історія створення об'єктно-орієнтованого підходу....................................................3
2. Пояснення об'єктно-орієнтованого підходу...............................................................4
3. Переваги та недоліки об'єктно-орієнтованого підходу.............................................5

3.1 Переваги об'єктно-орієнтованого підходу..........................................................5

3.2 Недоліки об'єктно-орієнтованого підходу..........................................................5

1. Принципи об'єктно-орієнтованого підходу................................................................7

4.1 Клас........................................................................................................................7

4.2 Об’єкт.....................................................................................................................7

4.3 Метод.....................................................................................................................7

4.4 Обмін повідомленнями........................................................................................7

4.5 Успадкування........................................................................................................ 8

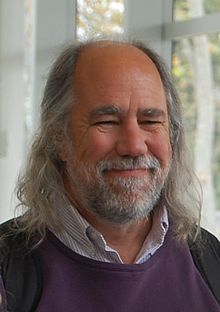
4.6 Приховування інформації.....................................................................................8

4.7 Абстрагування.......................................................................................................9

4.8 Поліморфізм..........................................................................................................9

## Історія створення підходу.

На основі структурного підходу сформувалася концепція об'єктно-орієнтованого методу (програмування, аналізу, проектування, баз даних) - фактично ціла філософія розробки систем та подання знань на базі потужного підходу. Основна відмінність об'єктно-орієнтованого аналізу і проектування від структурного полягає в декомпозиції проблеми на поняття (об'єкти), а не на функції. Історично розвиток цієї галузі почалося з об'єктно орієнтованого програмування. У 1980-1990-х рр. з'явилися об'єктно-орієнтовані методи розробки програмного забезпечення, запропоновані Граді Бучем (Booch), Джеймсом Рамбо (ОМТ - Object Modeling Technique) і Айваром Якобсоном (OOSE - Object Oriented Software Engineering).

**Граді Буч**

Американський вчений в галузі інформаційних технологій і програмування. Автор класичних праць з [об'єктно-орієнтованого аналізу](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1'%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7" \o "Об'єктно-орієнтований аналіз). Один з творців [UML](https://uk.wikipedia.org/wiki/UML" \o "UML).

**Рамбо, Джеймс**

[Американский](https://ru.wikipedia.org/wiki/%D0%A1%D0%A8%D0%90) [учёный в области информатики](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA" \o "Информатик) и объектной методологии, наиболее известный по своей работе над созданием [технологии объектного моделирования](https://ru.wikipedia.org/w/index.php?title=Object-modeling_technique&action=edit&redlink=1" \o "Object-modeling technique (страница отсутствует)) (OMT) и языка моделирования [UML](https://ru.wikipedia.org/wiki/UML" \o "UML).

**Якобсон, Ивар**

Шведский учёный в области информатики, внёсший вклад в развитие [UML](https://ru.wikipedia.org/wiki/UML" \o "UML), [RUP](https://ru.wikipedia.org/wiki/RUP" \o "RUP), [аспектно-ориентированного программирования](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Аспектно-ориентированное программирование).

Незважаючи на явну перевагу об'єктно-орієнтованих технологій розробки програмного забезпечення перед структурними, їх поширення було обмеженим, оскільки жоден з методів не давав єдиної і цілісної об'єктної моделі системи. Крім того, широкому розповсюдженню об'єктно-орієнтованих методів при розробці програмного забезпечення заважала відсутність єдиного стандарту мови об'єктно-орієнтованого моделювання.

Протягом 1994-1996 рр. вищезгадані розробники об'єднали свої зусилля під егідою Rational Software Corporation для створення єдиної мови моделювання, який втілив би всі істотні і успішні розробки в даній області і став би стандартом мови об'єктно-орієнтованого моделювання. Грандіозний працю, в якому поряд з Rational брали участь представники таких великих компаній, як Microsoft, IBM, Hewlett-Packard, Oracle, Platinum Technology і декількох сотень інших, завершився створенням в січні 1997 р версії 1.0 уніфікованої мови моделювання UML (Unified Modeling Language).

## Пояснення об'єктно-орієнтованого підходу.

Об'єктно-орієнтоване програмування - одна з [парадигм](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Парадигми програмування) [програмування](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Програмування), яка розглядає програму як множину [«об'єктів»](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1'%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Об'єкт (програмування)), що взаємодіють між собою. Основу ООП складають чотири основні концепції: [інкапсуляція](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F" \o "Інкапсуляція), [успадкування](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Успадкування (програмування)), [поліморфізм](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Поліморфізм (програмування)) та абстракція Однією з переваг ООП є краща [модульність](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Модуль (програмування)) 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](https://uk.wikipedia.org/wiki/1960-%D1%82%D1%96" \o "1960-ті) роках, вона не мала широкого застосування до [1990-х](https://uk.wikipedia.org/wiki/1990-%D1%82%D1%96" \o "1990-ті), 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](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Мова програмування) або підтримують ООП ([PHP](https://uk.wikipedia.org/wiki/PHP" \o "PHP), [Lua](https://uk.wikipedia.org/wiki/Lua" \o "Lua)) або ж є цілком [об'єктоорієнтованими](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1'%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Об'єктно-орієнтована мова програмування) (зокрема, [Java](https://uk.wikipedia.org/wiki/Java_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Java (мова програмування)), [C#](https://uk.wikipedia.org/wiki/C_Sharp_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "C Sharp (мова програмування)), [C++](https://uk.wikipedia.org/wiki/C++" \o "C++), [Python](https://uk.wikipedia.org/wiki/Python" \o "Python), [Ruby](https://uk.wikipedia.org/wiki/Ruby" \o "Ruby) і [Objective-C](https://uk.wikipedia.org/wiki/Objective-C" \o "Objective-C), [ActionScript 3](https://uk.wikipedia.org/wiki/ActionScript" \o "ActionScript), [Swift](https://uk.wikipedia.org/wiki/Swift_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Swift (мова програмування)), [Vala](https://uk.wikipedia.org/wiki/Vala" \o "Vala)).

На відміну від традиційних поглядів, коли програму розглядали як набір [підпрограм](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0" \o "Підпрограма), або як перелік [інструкцій](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D1%8F" \o "Інструкція) комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати [повідомлення](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D1%96%D0%BD_%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC%D0%B8" \o "Обмін повідомленнями), обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

## Переваги та недоліки об'єктно-орієнтованого підходу.

**3.1 Переваги об'єктно-орієнтованого підходу:**

* Повторне використання.

Це означає повторне використання деяких об'єктів, а не будівництво їх знову і знову. Це робиться з використанням класу. OOP дозволяє повторно використовувати існуючий код через успадкування. Ми можемо легко придбати існуючу функціональність та вдосконалити її, не переписуючи код знову. Це призводить до менш роздутого коду. Ми можемо використовувати його «n» кількість разів відповідно до наших потреб.

* Надлишок даних.

Це умова, створена в місці зберігання даних (можна сказати, Бази даних), коли однаковий фрагмент даних зберігається в двох окремих місцях. Тож надмірність даних є однією з найбільших переваг OOP. Якщо користувач хоче подібну функціональність у кількох класах, він / вона може продовжувати, записуючи загальні визначення класів для подібних функціональних можливостей та успадковуючи їх.

* **Краща продуктивність.**

З вищезазначеними фактами використання програми, безумовно, підвищує загальну продуктивність користувачів. Це призводить до більше виконаної роботи, досконалої програми, що має більше вбудованих функцій та легше читати, писати та підтримувати. Програміст OOP може зшивати нові програмні об'єкти для створення абсолютно нових програм. Чимала кількість бібліотек з великою кількістю корисних функцій робить це можливим.

* Технічне обслуговування коду

Ця функція є більш необхідною для будь-яких мов програмування, вона допомагає користувачам багато разів переробляти роботу. Підтримувати та змінювати існуючі коди завжди легко та економію часу, включаючи в нього нові зміни. Ведення коду простіше, оскільки легко додавати нові класи, об’єкти тощо без особливої перебудови чи змін.

* Гнучкість

ООП допомагає нам у гнучкому програмуванні, використовуючи функцію поліморфізму. Оскільки поліморфізм набуває різних форм, ми можемо мати оператори або функції, які працюватимуть із багатьма об’єктами, і таким чином позбавляти нас від запису різних функцій для кожного об’єкта.

**3.2 Недоліки об'єктно-орієнтованого підходу:**

* **Ускладнення методології.**

Застосування об'єктно-орієнтованого підходу вимагає введення додаткових способів подання інформації про предметну область і методів її аналізу. Мова UML включає більше 100 різних умовних позначень. Для успішного використання подібного механізму потрібна наявність певного рівня кваліфікації у фахівців. Для невеликих проектів більш ефективним може виявитися застосування класичних методів розробки. Розробка проектів, для яких найважливішим завданням є опис предметної області, і для яких неможливо знайти людину, яка розуміє цю предметну область в цілому також вимагає використання традиційних підходів, на увазі їх більшої доступності для неспеціалістів.

* **Складність реалізації.**

Об'єктно-орієнтовані проекти і їх програмна реалізації на об'єктно-орієнтованої мови, вимагають великих витрат часу і призводять до побудови більш складної і вимогливою до ресурсів програми, ніж класичні методи, які можуть виявитися більш ефективними для деяких завдань.

При об'єктно-орієнтованому підході в якості будівельних блоків використовуються об'єкти, що містять свої власні коди і дані. Структура програм при об'єктно-орієнтованому підході представляється графом взаємодії об'єктів, а не деревом ієрархії, як це має місце в структурному проектуванні.

## Принципи об'єктно-орієнтованого підходу.

### 4.1 Клас.

Клас визначає абстрактні характеристики деякої сутності, включно з характеристиками самої сутності (її атрибутами або властивостями) та діями, які вона здатна виконувати (її поведінкою, [методами](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Метод (програмування)) або можливостями). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять [модульність](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Модуль (програмування)) та [структурованість](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0" \o "Структура) в об'єктоорієнтовану 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.

### 4.2 Об'єкт.

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](https://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B0%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)&action=edit&redlink=1" \o "Стан (програмування) (ще не написана)). 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.

**4.3 Метод.**

Можливості об'єкта. Оскільки Сірко — Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба, щоб гавкав лише один окремий собака.

**4.4 Обмін повідомленнями.**

«Передача даних від одного процесу іншому, або надсилання викликів методів.»

Обмін повідомленнями — в [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Програмування), є способом координації в [конкурентних](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D1%96%D1%81%D1%82%D1%8C" \o "Конкуррентність), [паралельних](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%96_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D1%8E%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8" \o "Паралельні обчислювальні системи), та [об'єктно-орієнтованих](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1'%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Об'єктно-орієнтоване програмування) системах, та організації [взаємодії між процесами](https://uk.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D1%94%D0%BC%D0%BE%D0%B4%D1%96%D1%8F_%D0%BC%D1%96%D0%B6_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%B0%D0%BC%D0%B8" \o "Взаємодія між процесами). Координація робиться шляхом відсилання повідомлень отримувачу. Повідомлення можуть мати форму викликів функцій, сигналів, та пакетів даних. До найвідоміших моделей обчислень, основаних на обміні повідомленнями належать модель Актора та [числення процесів](https://uk.wikipedia.org/w/index.php?title=%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%96%D0%B2&action=edit&redlink=1" \o "Числення процесів (ще не написана)).

Функції відправлення та отримання повідомлень зазвичай мають назву відправити ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) send) та отримати ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) receive) відповідно. Вони є примітивами в системах обміну повідомленнями. Функція «відправити» має два параметри — адресу, на яку слі д надіслати повідомлення, та буфер даних повідомлення. Функція «отримати» також має два параметри — адреса звідки очікується повідомлення (може бути шаблоном) та буфер для збереження даних повідомлення.

За характеристиками, примітиви можна поділити на:

* Синхронні

Якщо примітиви відправити та отримати встановлюють попередній зв'язок між собою.

* Асинхронні

Примітив відправити називається асинхронним, якщо він повертає керування одразу після завершення зчитування даних з буфера даних повідомлення.

* Блокуючі

Примітив працює у блокуючому режимі, якщо він повертає керування лише після завершення виконання (як в асинхронному, так і в синхронному режимі).

не блокуючі

Примітив працює у не блокуючому режимі, якщо він одразу повертає керування, навіть коли повідомлення не зчитане або не отримане.

Таким чином, існує чотири варіанти відправки повідомлень:

* синхронний блокуючий,
* синхронний не блокуючий,
* асинхронний блокуючий,
* асинхронний не блокуючий.

Існує два варіанти для отримання повідомлень:

* синхронний блокуючий;
* синхронний не блокуючий.

**4.5 Успадкування (наслідування).**

Клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть навіть утворюватись цілі дерева успадкування. Наприклад, клас Собака може мати підкласи Коллі, Пекінес, Вівчарка тощо. Так, Сірко може бути екземпляром класу Вівчарка. Підкласи успадковують атрибути та поведінку своїх батьківських класів, і можуть вводити свої власні. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в Java дозволене лише одинарне успадкування, а в С++ і те й інше.

**4.6 Приховування інформації (інкапсуляція).**

Приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Так, наприклад, клас Собака має метод гавкати(). Реалізація цього методу описує як саме повинно відбуватись гавкання (приміром, спочатку вдихнути(), а потім видихнути() на обраній частоті та гучності). Петро, хазяїн пса Сірка, не повинен знати як він гавкає. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта.

Часто, члени класу позначаються як публічні ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) public), захищені ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) protected) та приватні ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) private), визначаючи, чи доступні вони всім класам, підкласам, або лише до класу в якому їх визначено. Деякі мови програмування йдуть ще далі: [Java](https://uk.wikipedia.org/wiki/Java_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Java (мова програмування)) використовує ключове слово private для обмеження доступу, що буде дозволений лише з методів того самого класу, protected — лише з методів того самого класу і його нащадків та з класів із того ж самого пакету, [C#](https://uk.wikipedia.org/wiki/C_Sharp_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "C Sharp (мова програмування)) та [VB.NET](https://uk.wikipedia.org/w/index.php?title=Visual_Basic.NET&action=edit&redlink=1" \o "Visual Basic.NET (ще не написана)) відкривають деякі члени лише для класів із тієї ж збірки шляхом використання ключового слова internal (C#) або Friend (VB.NET), а [Eiffel](https://uk.wikipedia.org/wiki/Eiffel_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)" \o "Eiffel (мова програмування)) дозволяє вказувати які класи мають доступ до будь-яких членів.

**4.7 Абстрагування.**

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі — як Коллі і як Тварина (можливо, батьківський клас Собака) під час підрахунку тварин Петра.

**4.8 Поліморфізм**

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. Наприклад, якщо Собака отримує команду голос(), то у відповідь можна отримати Гав; якщо Свиня отримує команду голос(), то у відповідь можна отримати Рох-рох. На практиці — це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. Залежно від того, що передається, і вибирається відповідна підпрограма.

## Висновок

Об'єктно-орієнтоване програмування спрощує процес програмування для нас. Він має багато значень, таких як повторне використання, ефективність та підтримка коду. Хоча спочатку може бути важко зрозуміти поняття OOP.

Об'єктно-орієнтоване програмування – це інструмент. Відповідно, як будь-який інструмент, його можна застосовувати з різним ступенем ефективності, за призначенням або за призначенням. Питання ефективного застосування відносяться до іншої дисципліни – об'єктно-орієнтованого проектування та аналізу.

Розуміння концепцій та базових елементів ООП дозволяє бачити у тих чи інших особливостях мов програмування не заявлені маркетологами «унікальні можливості», а різновиди реалізації широко відомих ідей.

## Література.

1. [Об'єктно-орієнтоване моделювання[Wikipedia].](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \l ":~:text=%D0%9E%D0%B1'%D1%94%D0%BA%D1%82%D0%BD%)
2. [Об'єктно-орієнтоване програмування.[Wikipedia]](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F).
3. [Об'єктно-орієнтований підхід до програмування[Портал Знань].](http://www.znannya.org/?view=csharp-oop)
4. [ОБ'ЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД[stud.com.ua].](https://stud.com.ua/174121/informatika/obyektno_oriyentovaniy_pidhid)
5. [Граді Буч[Wikipedia].](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D0%B4%D1%96_%D0%91%D1%83%D1%87)
6. [Якобсон, Ивар[Wikipedia].](https://ru.wikipedia.org/wiki/%D0%AF%D0%BA%D0%BE%D0%B1%D1%81%D0%BE%D0%BD,_%D0%98%D0%B2%D0%B0%D1%80)
7. [Рамбо, Джеймс[Wikipedia].](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%BC%D0%B1%D0%BE,_%D0%94%D0%B6%D0%B5%D0%B9%D0%BC%D1%81)
8. [Обмін повідомленнями[Wikipedia].](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D1%96%D0%BD_%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC%D0%B8)
9. [Переваги та недоліки об'єктно-орієнтованого підходу[studwood.net].](https://studwood.net/1586519/informatika/perevagi_nedoliki_obyektno_oriyentovanogo_pidhodu)