Лекція 11

ОБ’ЄКТНО ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

11.1 Основні аспекти об’єктно орієнтованого програмування

Об'єктно-орієнтоване програмування (ООП) має дуже широкий вплив та застосування, тому що воно працює на декількох рівнях і забезпечує швидку та ефективну розробку та технічне обслуговування. За ООП дотримуються висхідного підходу до проектування застосунків знизу вгору.

Метод висхідної розробки (знизу - вгору) полягає в наступному. Спочатку будується модульна структура програми у виді дерева. Потім по черзі програмуються модулі програми, починаючи з модулів самого нижнього рівня (листи дерева модульної структури програми), у такому порядку, щоб для кожного програмного модуля були вже запрограмовані всі модулі, до яких він може звертатися. Після того, як усі модулі програми запрограмовані, виконується їхнє почергове тестування і налагодження у такому ж (висхідному) порядку, у якому велося їхнє програмування.

Висхідне проектування починається з виявлення елементів даних, які потім групуються в набори даних. Спочатку визначаються атрибути (властивості об’єктів), які потім об'єднуються в сутності. Висхідне проектування включає операції синтезу, що передбачає виконання компоновки із заданої множини функціональних залежностей між об'єктами предметної області вихідних відношень.

Цей підхід рекомендується застосовувати у тому випадку, якщо розробляється невеликий застосунок з незначною кількістю об'єктів, їх властивостей і транзакцій.

Слово об'єктно-орієнтоване – це поєднання двох слів: об'єкт (object) і орієнтований (oriented). Семантичне значення терміна «об'єкт» – це предмет або сутність, що існує в реальному світі. Значення терміна «орієнтований» - це цікавленість в певному виді предметів або сутностей. В загальному визначені ООП – це шаблон програмування, що описує об'єкти або сутності реального світу, їх властивості та операції, які можна виконувати над цими об’єктами.

Об'єктно-орієнтоване програмування – це концепція проектування комп'ютерного програмування або методологія, яка організовує/ моделює проектування програмного забезпечення навколо даних, або об'єктів, а не функцій і логіки.

Об'єкт називають полем даних, яке має унікальні атрибути і поведінку. Все, що розглядається в ООП згруповано як окремі автономні об'єкти.

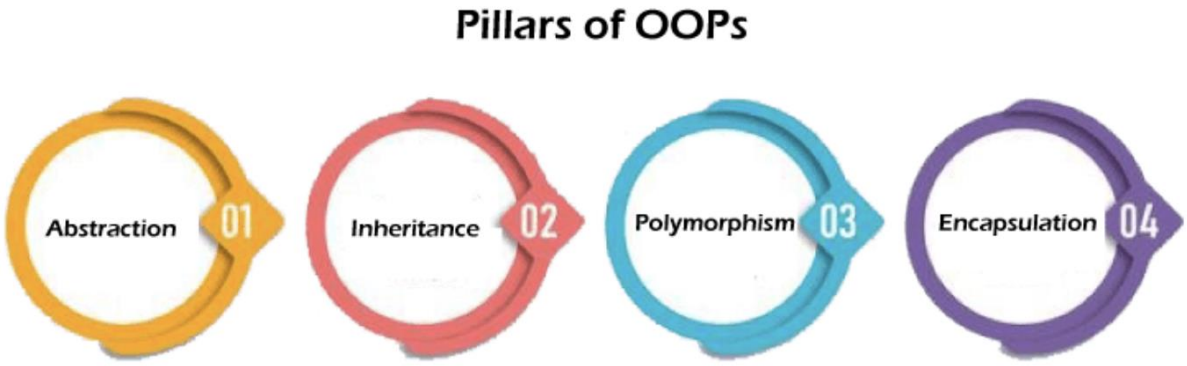
ООП є найпопулярнішою моделлю програмування серед розробників. Така концепція добре підходить для програм, які є великими, складними, і активно оновлюються або підтримуються. ООП спрощує розробку та обслуговування програмного забезпечення, надаючи основні концепції, такі як абстракція, спадковість, поліморфізм та інкапсуляція.

Особливості концепції ООП:

- Всі елементи предметної області є об’єктами.
- Розробник маніпулює об'єктами, які обмінюються повідомленнями.
- Кожен об'єкт є екземпляром класу.
- Клас містить атрибут і поведінку, пов’язані з об’єктами.

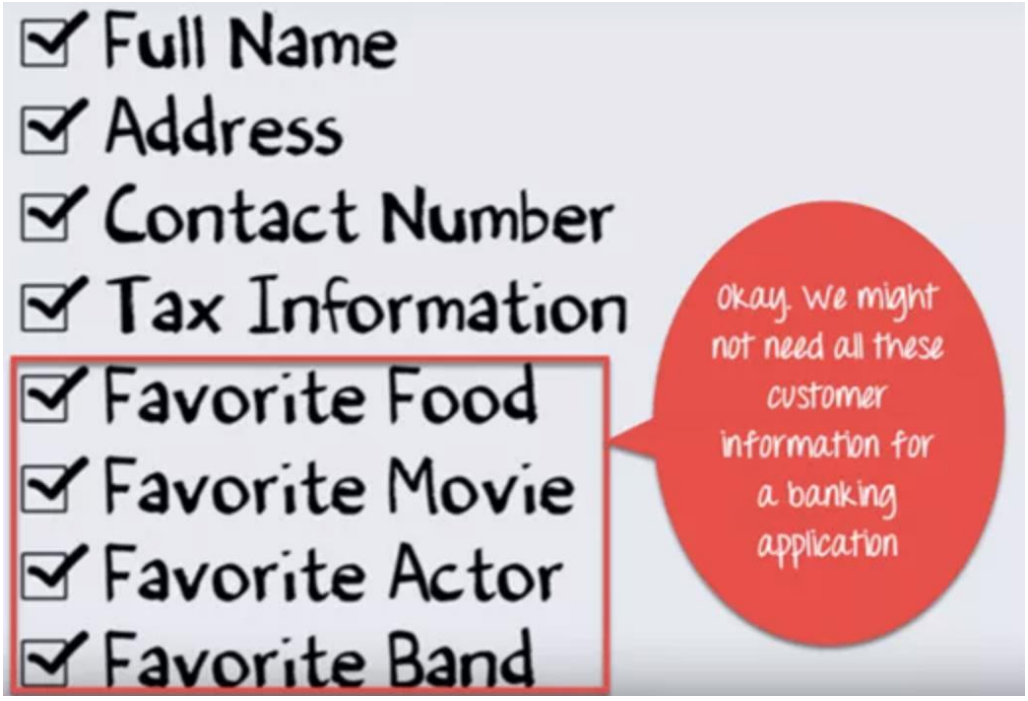
В основну ООП покладено такі концепції:

1 абстракція;
2 інкапсуляція;
3 успадкування;
4 поліморфізм.



Розглянемо кожен з механізмів детальніше.

Абстракція – це концепція ООП, яка «показує» лише необхідні атрибути та «приховує» непотрібну інформацію. Основна мета абстракції – приховування непотрібних деталей від користувачів. Абстракція – це вибір даних із більшого пулу, щоб показати користувачеві лише відповідні деталі об’єкта. Такий підхід допомагає зменшити складність та зусилля програмування. Це одна з найважливіших концепцій ООП.



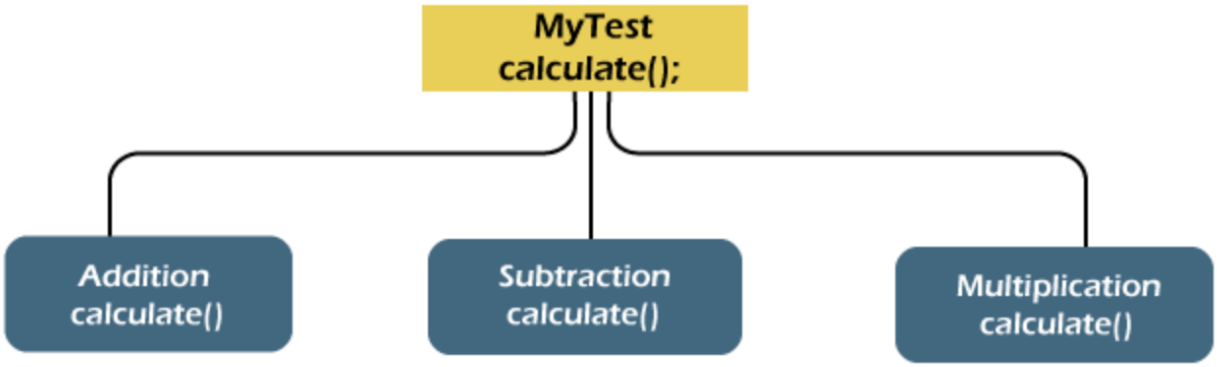
Існують такі переваги абстракції:

1 Це зменшує складність ПЗ (ще на етапі розробки моделі).
2 Знижує ризик помилок під час роботи з ПЗ.
3 Облегшує ведення ПЗ.
4 Підвищити безпеку та конфіденційність.

Абстрактний клас – це тип класу в ООП, який оголошує один або кілька абстрактних методів. Ці класи можуть мати як абстрактні методи, так і конкретні методи. Звичайний клас не може мати абстрактних методів. Абстрактний клас - це клас, який містить принаймні один абстрактний метод.

Абстрактний метод – це метод, який має лише визначення методу, але не містить реалізації. Метод без тіла відомий як абстрактний метод. Це має бути оголошено в абстрактному класі. Абстрактний метод ніколи не буде

остаточним, оскільки абстрактний клас повинен реалізовувати всі абстрактні методи.



Основною перевагою використання абстракції в програмуванні є те, що вона дозволяє згрупувати кілька пов’язаних класів як братів і сестер.

Абстракція в об’єктно-орієнтованому програмуванні допомагає зменшити складність процесу проектування та впровадження програмного забезпечення.

Абстрактні методи в основному оголошуються там, де два або більше підкласів також роблять одне і те ж по-різному за допомогою різних реалізацій. Він також розширює той самий клас Abstract і пропонує різні реалізації абстрактних методів.

Абстрактні класи допомагають описати загальні типи поведінки та ієрархію об’єктно-орієнтованого програмування класів. Він також описує підкласи, щоб запропонувати деталі реалізації абстрактного класу.

Різниця між абстракцією та інкапсуляцією

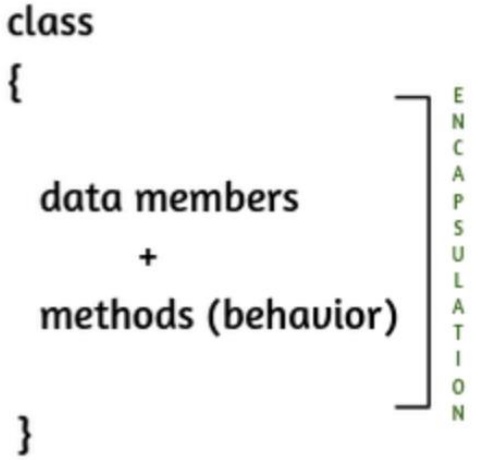
Абстракція	**Інкапсуляція**
Абстракція в об’єктно-орієнтованому програмуванні вирішує проблеми на рівні проектування.	Інкапсуляція вирішує рівень її реалізації.
Абстракція в програмуванні полягає у приховуванні небажаних деталей, одночасно показуючи найважливішу інформацію.	Інкапсуляція означає зв’язування коду та даних в одну одиницю.

Абстракція даних дозволяє зосередити увагу на тому, що повинен містити інформаційний об'єкт	Інкапсуляція означає приховування внутрішніх деталей або механіки того, як об’єкт щось робить з міркувань безпеки.

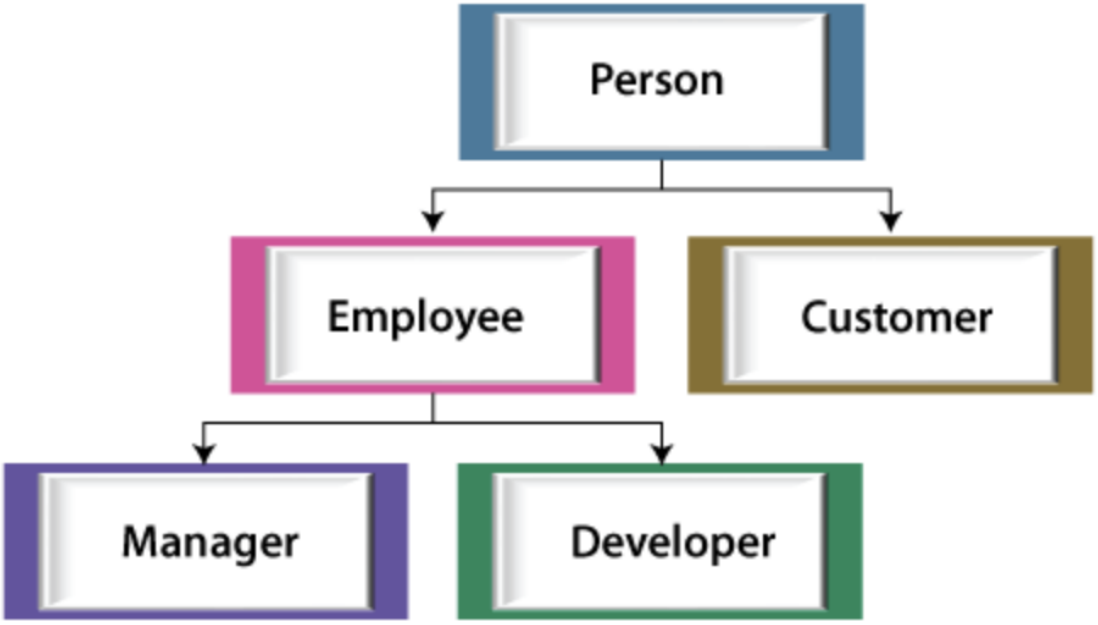
Різниця між абстрактним класом та інтерфейсом

Анотація класу	**Інтерфейс**
Абстрактний клас може мати як абстрактні, так і не абстрактні методи.	Інтерфейс може мати лише абстрактні методи.
Він не підтримує множинне успадкування.	Він підтримує кілька видів спадкоємства.
Він може забезпечити реалізацію інтерфейсу.	Він не може забезпечити реалізацію абстрактного класу.
Абстрактний клас може мати захищений та абстрактний загальнодоступні методи.	Інтерфейс може мати лише загальнодоступні абстрактні методи.
Абстрактний клас може мати фінальну, статичну або статичну кінцеву змінну з будь-яким специфікатором доступу.	Інтерфейс може мати лише загальнодоступну статичну кінцеву змінну.

Інкапсуляція – це механізм, який дозволяє зв'язувати дані та функції класу в єдину сутність. Він захищає дані та функції від зовнішнього втручання та неправильного використання, а тому також є механізмом забезпечення безпеки даних. Найкращим прикладом інкапсуляції є клас.



Спадкування. Концепція дозволяє новоствореним (дочірнім) класам успадковувати або набувати властивостей вже існуючого (батьківського) класу. Такий механізм відомий як спадкування. Він забезпечує повторне використання коду.



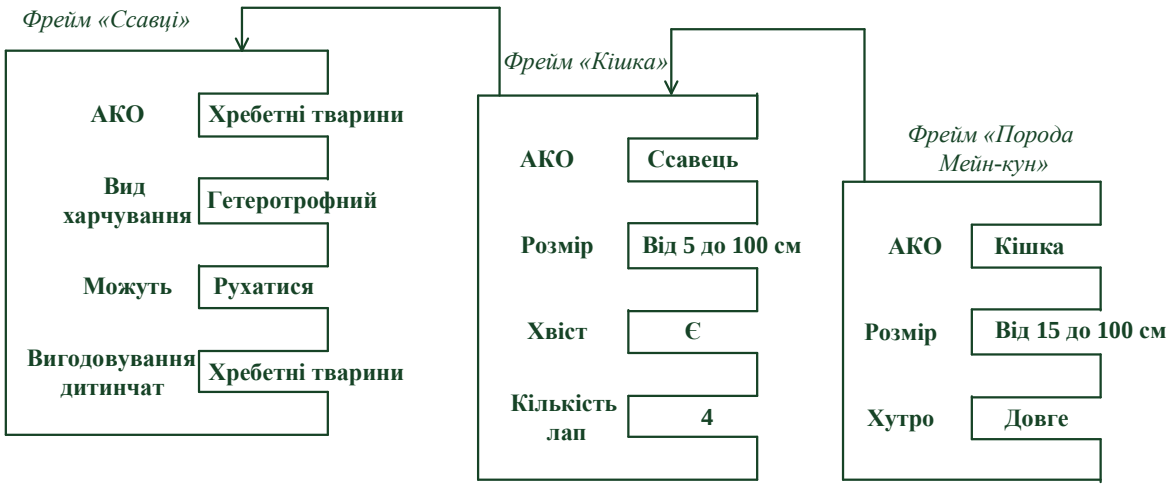
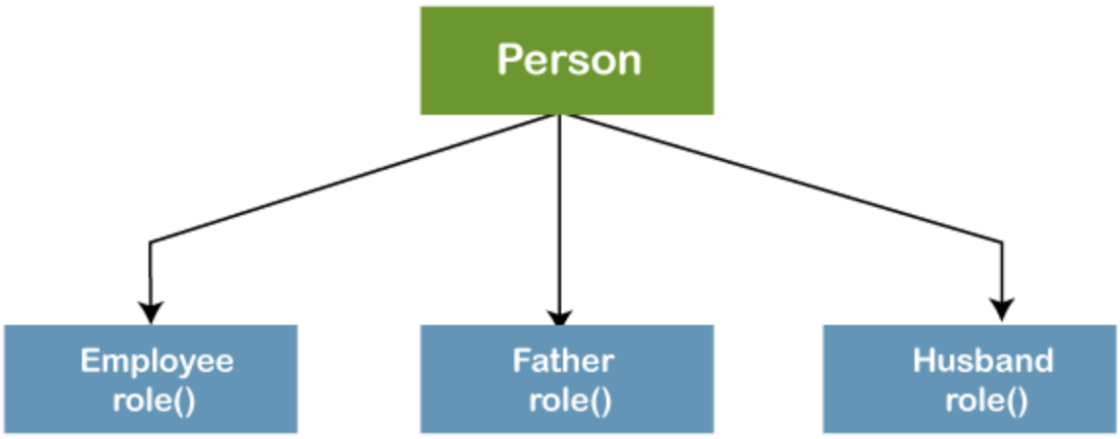


Рисунок 11.1 – Мережа фреймів з наслідуванням властивостей класу

Спадкування, це такий механізм, який дозволяє розширювати клас за рахунок методів іншого класу.

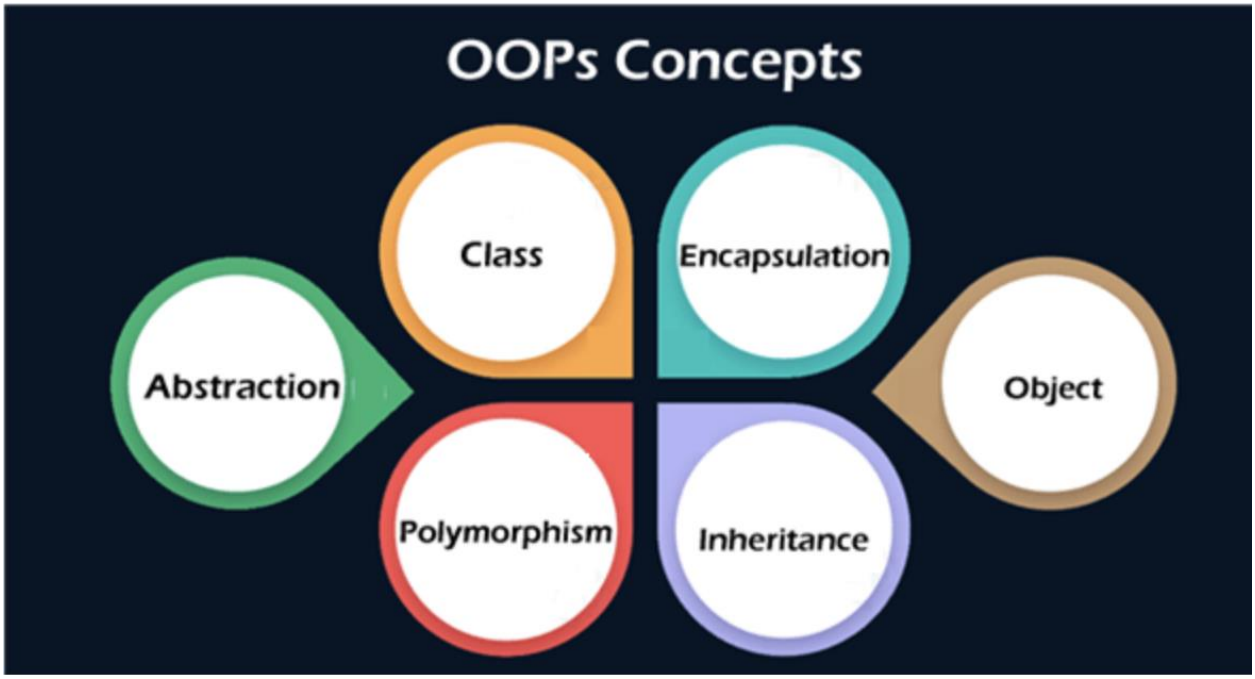
Поліморфізм. Слово поліморфізм походить від двох слів: *poly* – багато та *morphs* – форми. Цей механізм дозволяє створювати методи з тією ж назвою,

але різними сигнатурами (підписами) методу. Він дозволяє розробнику створювати чистий, розумний, зрозумілий і стійкий код.



Наведена вище фігура найкраще описує поняття поліморфізму. Кожна людина відіграє різні ролі в різних соціальних групах: роль працівника в офісі, батька і чоловіка – в родині.

Крім наведених механізмів концепція ООП включає такі поняття як «об’єкт» та «клас».



Об'єкт – сутність (фізична/матеріальна або логічна/ідеальна/ нематеріальна) реального світу, який має атрибути, поведінку та властивості.

Об’єкт є екземпляром певного класу, який містить функції та змінні, які визначаються в класі. В пам'яті під об’єкт резервується певний простір. Різні об'єкти мають різні стани або атрибути та поведінку.

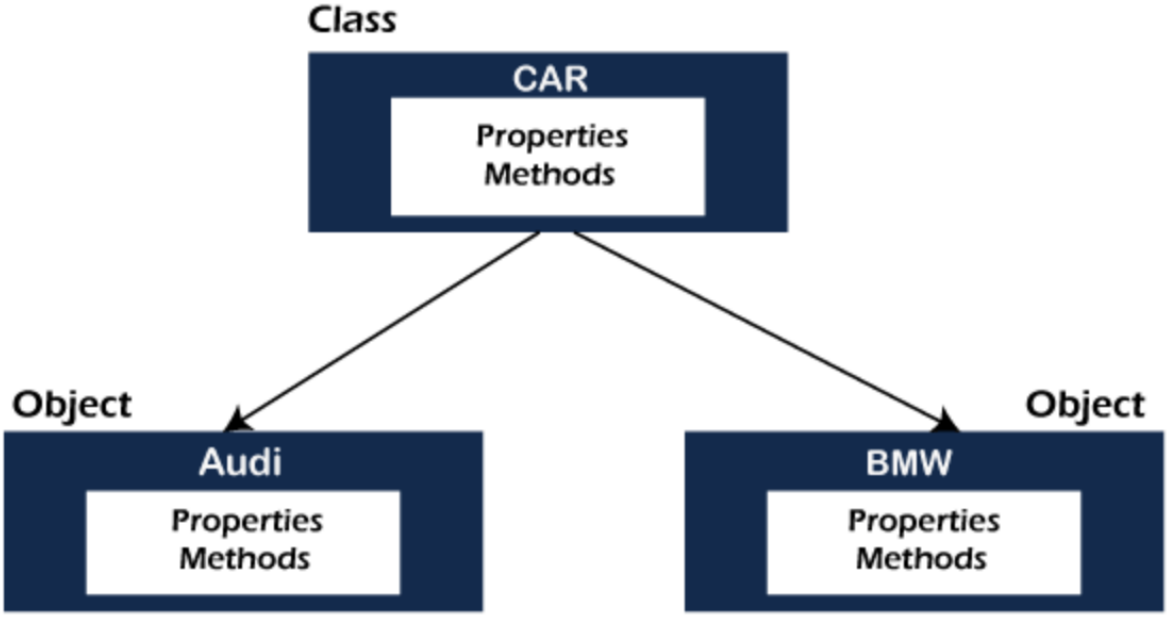
Для об’єкта є характерними:

– Стан: представляє дані (значення) об’єкта.

– Поведінка: представляє поведінку (функціонал) об’єкта, наприклад депозит, зняття тощо.

– Ідентифікація: реалізується через унікальний ідентифікатор, значення якого не видно зовнішньому користувачеві, однак він використовується внутрішньо для однозначної ідентифікації кожного об’єкта.

Клас – це схема (логічна сутність) або шаблон об'єкта. Це визначений користувачем тип даних. Усередині класу визначаються змінні, константи, функції та методи. Клас об’єднує дані та функції в єдиний блок. Пам'ять під час створення класу не резервується. Клас може існувати без об’єкта, але навпаки неможливо.



Зв’язування (Coupling)

У програмуванні поділ проблем з подальшим пов’язуванням модулів для її вирішення відомий як зчеплення. Це означає, що об’єкт не може напряму змінювати чи модифікувати стан або поведінку інших об’єктів. Відповідно до

того наскільки тісно з’єднані два об’єкти виокремлюють два типи зв’язування: слабе зв’язування та жорстке зв’язування.

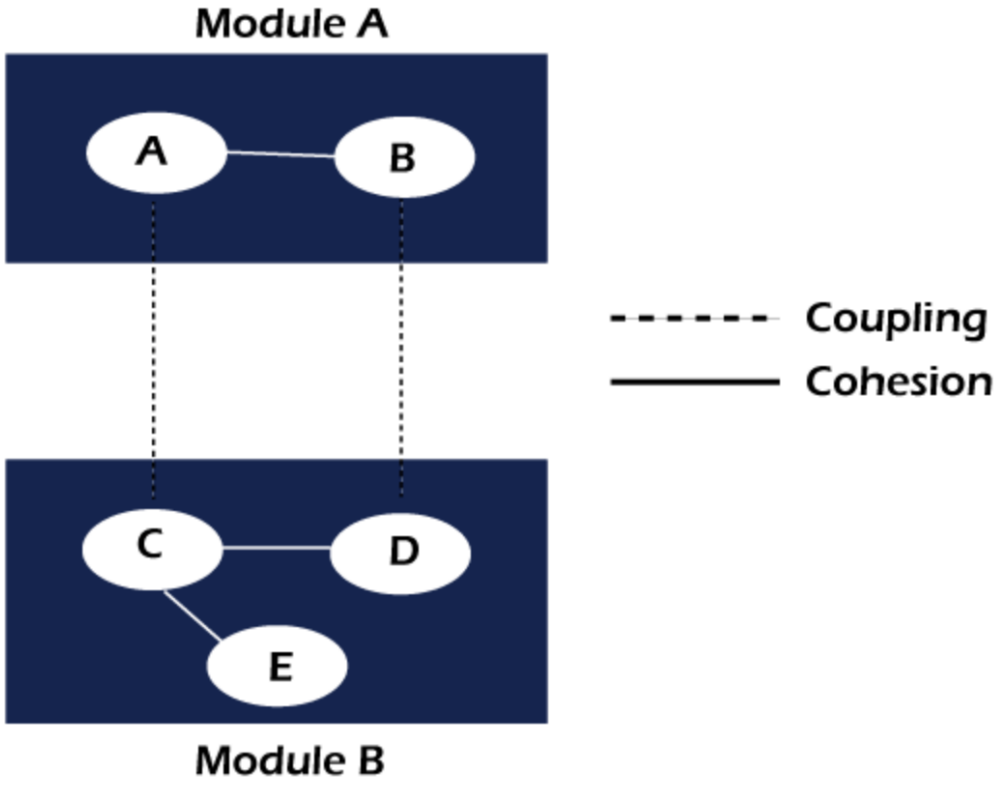
Об’єкти, які є незалежними один від одного і безпосередньо не змінюють стан інших об’єктів, називаються слабо зв’язаними. Слабкий зв’язок робить код більш гнучким, придатним до модифікації, з ним легше працювати.

Об’єкти, які залежать від інших об’єктів і можуть змінювати стани інших об’єктів, називаються жорстко зв’язаними. Жорстке зв’язування створює умови, за яких модифікація коду одного об’єкта вимагає зміни коду інших об’єктів. Повторне використання коду в цьому випадку, оскільки не можливо розділити код на окремі модулі.

Згуртованість (Cohesion)

В ООП згуртованість означає ступінь сумісності елементів всередині модуля. Згуртованість показує міцність зв’язку між модулем і даними. Ця властивість є протилежною до попередньої.

Чим вища зв'язність модуля або класу, тим краще об'єктно-орієнтований дизайн.



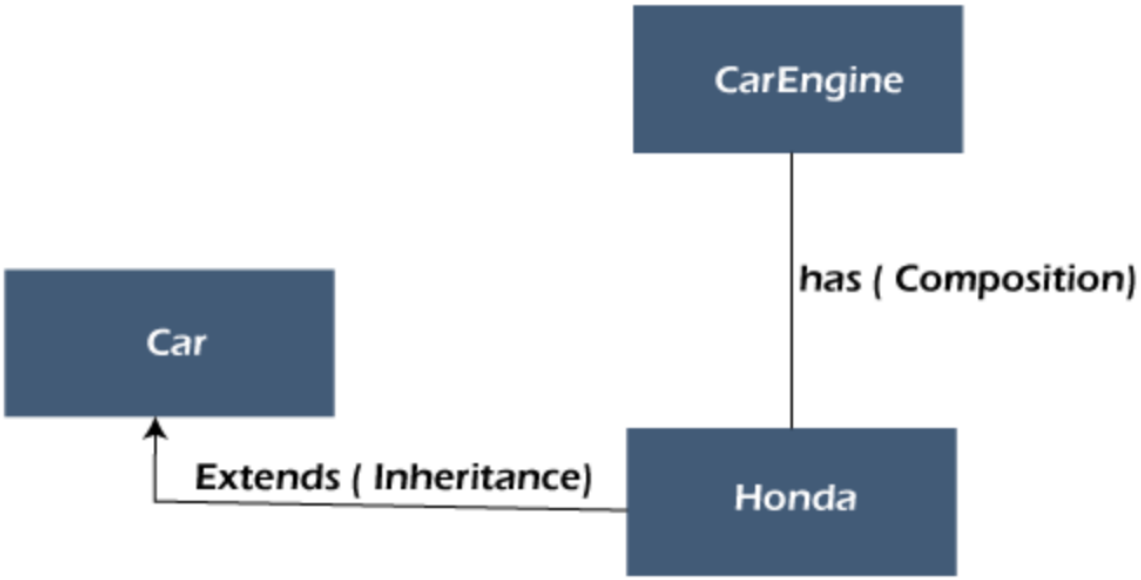
Існує два типи згуртованості: висока та низька.

Висока згуртованість пов'язана з декількома необхідними якостями програмного забезпечення, включаючи сталість, надійність і зрозумілість.

Низька згуртованість пов'язана з небажаними якостями: ускладнення модифікації та тестування, повторного використання та погана читабельність.

Композиція

Композиція є однією з важливіших понять в ООП. Він описує клас, який посилається на один або кілька об'єктів інших класів, наприклад, змінних. Це дозволяє нам моделювати між об'єктами асоціації типу has-a. Прикладом такої асоціації в реальному світі є автомобіль, який має (has-a) двигун.



В загальному випадку використовуються наступні види відношень:

1) «частина-ціле» (*ієрархічні* відносини, що визначають підклас класу або елемент множини);

2) «виробляє», «впливає» (*функціональні* зв’язки, які визначаються подібними дієсловами);

3) «більше», «менше», «дорівнює», «небільше», «неменше» (*кількісні* відносини, що позначають порівняння за кількісними показниками);

4) «далеко від», «близько до», «за», «під», «над» (*просторові* відносини);

5) «раніше», «пізніше», «протягом» (*часові* відносини);

6) «мати властивість», «мати значення» (*атрибутивні* відносини);

7) «та», «або», «ні», «тоді і тільки тоді» (*логічні* відносини).

Композиція має такі переваги:

- повторне використання існуючого коду;
- розробка за зрозумілих API;
- модифікація реалізації класу, який використовується в композиції, без адаптації зовнішніх сутностей.

Асоціації

Асоціація визначає зв'язок між об'єктами. Зверніть увагу, що об’єкт може бути пов’язаний з одним або декількома об’єктами. Відносини можуть бути односпрямованими або мати два напрямки спрямування. Існують наступні види асоціації.

Один-до-одного (One to One): людина-паспорт громадянина;

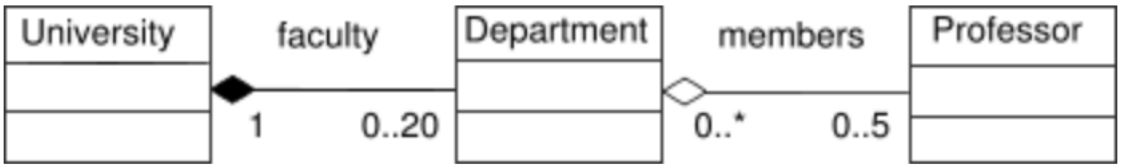
Один-до-багатьох (One to Many): група-здобувач;

Багато-до-одного (Many to One): група-факультет;

Багато-до-багатьох (Many to Many): здобувач-дисципліна.

Агрегація – методика створення нового класа з вже існуючих класів шляхом їх включення. Це вдосконалена форма асоціації, за якої кожен об'єкт має свій життєвий цикл і може існувати незалежно від «батьківського» класу. Це ще один спосіб повторного використання об'єктів. Приклад агрегації: автомобіль має двигун, двері тощо.

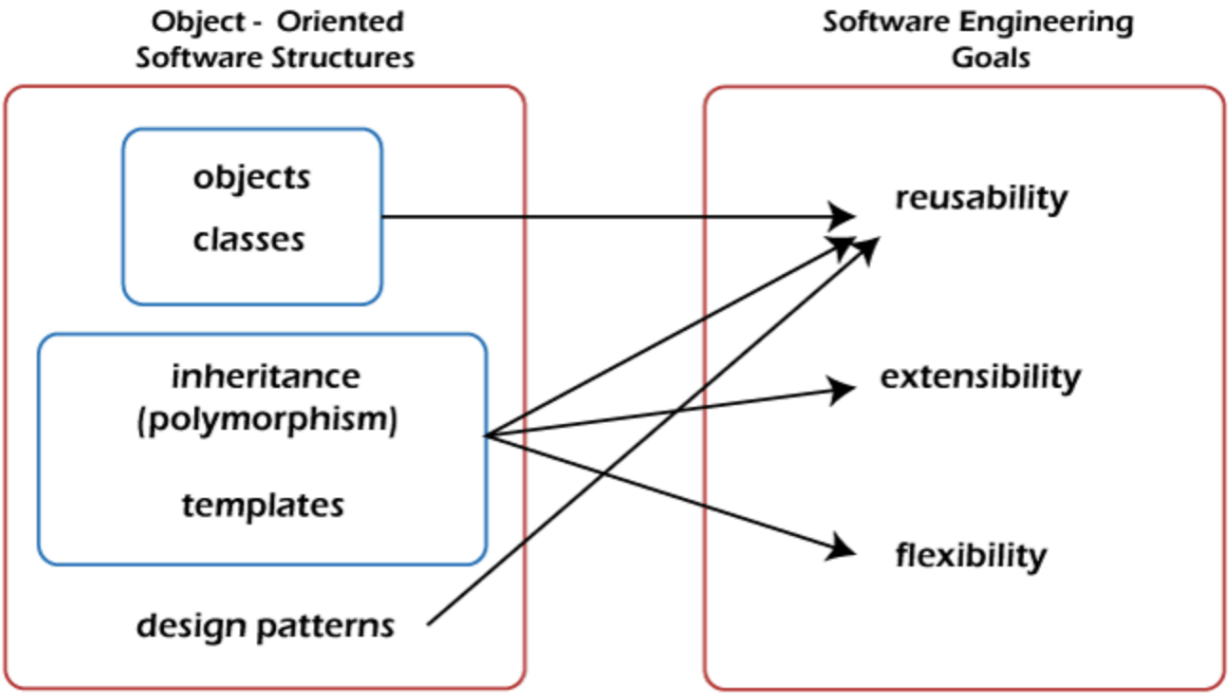
Інший приклад:



Агрегація: викладачі – факультети, викладачі залишаються існувати після знищення факультета

Композиція: університет – факультети, факультети без університета існувати не можуть.

Висновок: ООП є еволюційним розвитком розробки програмного забезпечення. Використання ООП у розробці програмного забезпечення є хорошою практикою, оскільки воно досягає трьох основних цілей розробки ПЗ – забезпечення: багаторазового використання коду; розширення можливостей коду; гнучкість та адаптивність (модифікованість) коду.



Перевагами ООП є:

– розробка модульного, масштабованого, розширюваного, багаторазового коду легкого у супроводі;

– представлення складної проблеми у вигляді структури простих задач;

– можливість використання об’єкта в різних модулях ПЗ;

– можливість легко змінювати або додавати код, не впливаючи на інші блоки коду;

– забезпечення безпеки завдяки функціям інкапсуляції та приховування даних;

– ефективність колективної розробки великих та складних проектів;

– можливість легкого налагодження та тестування.

Обмеження та недоліки ООП:

– вимагає інтенсивних процесів тестування;

– вирішення проблем займає більше часу порівняно з процедурно-орієнтованим програмуванням;

– розмір програм, створених за ООП може бути більшим ніж у програм, написаних за допомогою процедурно-орієнтованого підходу програмування;

– розроблення ПЗ з використанням ООП потребує значного обсягу попередньої роботи та планування;

– ООП-код важко зрозуміти, якщо у вас немає відповідної документації класів;

– у деяких випадках ці програми можуть споживати великий обсяг пам’яті;

– не підходить для дрібних проблем.

Відповідно до індексу TIOBE двадцятка найкращих ООП мов: Java , C++ , C# , Python , R , PHP , Visual Basic.NET , JavaScript , Ruby , Perl , Object Pascal, Objective-C , Dart , Swift , Scala , Kotlin , Common Lisp, MATLAB і Smalltalk.

TIOBE programming community index – індекс, що оцінює популярність мов програмування шляхом підрахунку результатів пошукових запитів, які містять назву мови (запит вида +"<language> programming").

Сферами застосування ООП є розробка:

1 Програм комп’ютерної графіки
2 Об'єктно-орієнтованих БД
3 Дизайна інтерфейсу користувача, наприклад вікон
4 Систем реального часу
5 Симуляцій та моделювання
6 Систем клієнт-сервер
7 Систем штучного інтелекту
8 Програмного забезпечення CAD/CAM
9 Систем автоматизації діловодства

11.2 Об’єкти у JavaScript

Об’єкт JavaScript — це сутність, яка має стан і поведінку (властивості та метод). Наприклад: автомобіль, ручка, велосипед, стілець, скло, клавіатура, монітор тощо.

JavaScript – це об’єктно-орієнтована мова. У JavaScript усе є об’єктом.

JavaScript базується на шаблонах, а не на класах. Тут ми не створюємо клас, щоб отримати об’єкт. Але ми безпосередньо створюємо об’єкти.

Існує три способи створення об’єктів у JavaScript:

1 За літералом об’єкта.
2 Шляхом безпосереднього створення екземпляра Object (за допомогою нового ключового слова).
3 За допомогою конструктора об’єктів (за допомогою нового ключового слова).

1) Створення об’єкта за літералом

Створення об’єкта за допомогою об’єктного літералу має наступний синтаксис:

```
object={property1:value1,property2:value2.....propertyN:valueN}
```

Зауваження: Властивість і значення розділені двокрапкою.

Розглянемо простий приклад створення об’єкта в JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Результат виконання наведеного вище прикладу наступний:

```
102 Shyam Kumar 40000
```

2) Створення об’єкта через додання екземпляра

Синтаксис безпосереднього створення об'єкта наведено нижче:

```
var objectname=new Object();
```

Тут ключове слово *new* використовується для створення об’єкта.

Розглянемо приклад безпосереднього створення об'єкта.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

Результат виконання наведеного вище прикладу наступний:

```
101 Ravi 50000
```

3) Створення об’єктів за допомогою конструктора об’єктів

За використання цього методу потрібно створити функцію з аргументами. Кожне значення аргументу можна призначити в поточному об’єкті за допомогою ключового слова.

Ключове слово *this* посилається на поточний об’єкт.

Приклад створення об'єкта конструктором об'єктів наведено нижче.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

Результат виконання наведеного вище прикладу наступний:

```
103 Vimal Jaiswal 30000
```

11.3 Визначення методу в об’єкті JavaScript

Можна визначити метод в об’єкті JavaScript. Але перед визначенням методу потрібно додати властивість у функцію з тим самим ім’ям, що й метод.

Нижче наведено приклад визначення методу в об’єкті.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;

this.changeSalary=changeSalary;
function changeSalary(otherSalary){
this.salary=otherSalary;
}
}
e=new emp(103,"Sonoo Jaiswal",30000);
document.write(e.id+" "+e.name+" "+e.salary);
e.changeSalary(45000);
document.write("<br>"+e.id+" "+e.name+" "+e.salary);
</script>
```

Результат виконання наведеного вище прикладу наступний:

```
103 Sonoo Jaiswal 30000
103 Sonoo Jaiswal 45000
```

У таблиці нижче наведено методи Object, які застосовують у JS

N	Метод	Опис
1	Object.assign()	Використовується для копіювання перелічуваних і власних властивостей з вихідного об’єкта в цільовий об’єкт
2	Object.create()	Використовується для створення нового об’єкта з указаним прототипом об’єкта та властивостями.
3	Object.defineProperty()	Використовується для опису деяких поведінкових атрибутів властивості.
4	Object.defineProperties()	Використовується для створення або налаштування кількох властивостей об’єкта.
5	Object.entries()	Повертає масив із масивами пар ключів і значень.
6	Object.freeze()	Запобігає видаленню існуючих властивостей.
7	Object.getOwnPropertyDescriptor()	Повертає дескриптор властивості для вказаної властивості вказаного об’єкта.
8	Object.getOwnPropertyDescriptors()	Повертає всі власні дескриптори властивостей заданого об’єкта.
9	Object.getOwnPropertyNames()	Повертає масив усіх знайдених властивостей (перелічуваних чи ні).
10	Object.getOwnPropertySymbols()	Повертає масив усіх властивостей ключа символу.
11	Object.getPrototypeOf()	Повертає прототип зазначеного об'єкта.
12	Object.is()	Визначає, чи є два значення однаковими.
13	Object.isExtensible()	Визначає, чи є об’єкт розширюваним
14	Object.isFrozen()	Визначає, чи був об’єкт заморожений.
15	Object.isSealed()	Визначає, чи запечатаний об’єкт.
16	Object.keys()	Повертає масив імен властивостей даного об’єкта.

17	Object.preventExtensions()	Використовується для запобігання будь-яким розширенням об’єкта.
18	Object.seal()	Запобігає додаванню нових властивостей і позначає всі наявні властивості як такі, що не підлягають налаштуванню.
19	Object.setPrototypeOf()	Встановлює прототип зазначеного об’єкта на інший об’єкт.
20	Object.values()	Повертає масив значень.

https://www.javatpoint.com/javascript-objects