# Об'єктно-орієнтовне моделювання та проектування складних систем

Лекція 4 – Діаграма класів

Ніколаєнко Дмитро Володимирович

# Зміст

- Що таке клас
- Рівні діаграми класів
- Діаграма класів
- Атрибути класів
- Операції класів
- Відношення між класів

# Вступ

**Клас** – множина об’єктів, що мають спільну структуру, властивості та поведінку.

# Рівні діаграми класів

# Рівні діаграми класів

Концептуальна (аналітична) модель

Модель проектування

Модель реалізації

# Концептуальна (аналітична) модель

- Описує бізнес-модель
- Відображає тільки класи прикладних об'єктів
- Використовується для аналізу
- Відповідає вимогам метамоделі
- Головна вимога - концептуальна цілісність

# Модель проектування

- Застосовується при проектуванні інформаційних систем
- Призначена для подальшої розробки моделей реалізації
- Містить основні атрибути класів які потрібні для розуміння реалізації
- Відповідає вимогам нотації проте може мати опущені елементи
- Головна вимога - зрозумілість

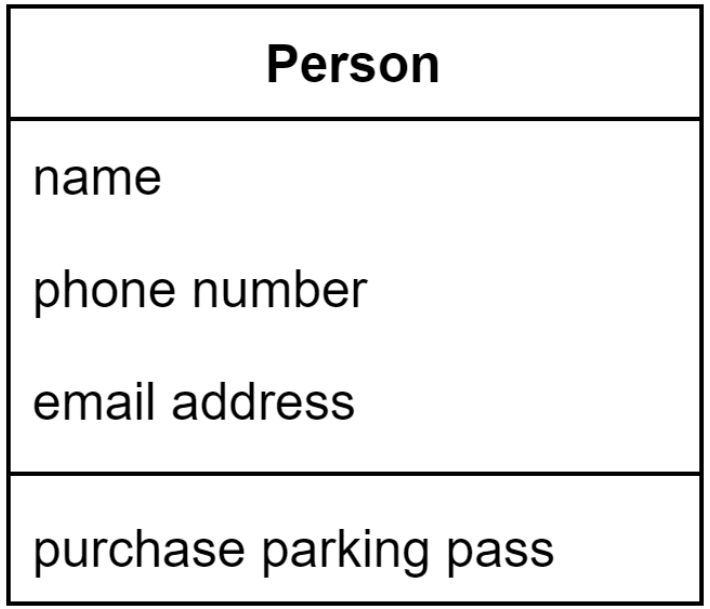
# Модель реалізації

- Містить класи, які використовуються безпосередньо у програмному коді
- Містить всі необхідні деталі класу (властивості, методи)
- Викладена у відповідності до вимог нотації
- Головна вимога - повнота

# Діаграма класів

# Діаграма класів

- Поділяється на 3 секції:
- **Назва** (напівжирним, по центру, починається з великої літери)
- **Атрибути** (по лівому краю, перша літера мала)
- **Операції** (по лівому краю, перша літера мала)



# Діаграма класів

**Назва класу** є обов'язковим елементом.

Назва має бути **унікальною** в межах моделі.

Клас – це **не таблиця** в базі даних

Клас – це сутність яка **має поведінку** (в більшості випадків)

Клас – це сутність, яка поєднує інші об'єкти за **спільними властивостями** та поведінкою

На початкових етапах аналізу достатньо вказувати лише назву класу

З ходом пропрацювання моделі класи наповнюються атрибутами та методами

# Атрибути класів

# Атрибути

Атрибут описує властивості у вигляді тексту в першому блоці класу.

Повна форма виглядає так:

```
Видимість назва: тип [кратність] = значення за замовченням {властивості}
```

Наприклад:

```
- username: String [1] = “Noname” {readOnly}
```

# Видимість атрибутів

Характеризує доступність цього атрибуту з інших класів системи

\+ - **відкритий** (public) – такий атрибут доступний з будь-якого іншого класу, пакета або модуля

\- - **закритий** (private) – такий атрибут доступний тільки з цього класу

\# - **захищений** (protected) – такий атрибут доступний тільки з підкласів цього суперкласу (при використанні спадкування)

Видимість атрибуту не є обов’язковою і може бути пропущена

# Кратність атрибутів

Характеризує загальну кількість конкретних атрибутів даного типу, що входять до складу окремого класу

[0..1] - кратність атрибута може набувати значення 0 або 1. При цьому 0 означає відсутність значення для даного атрибута.

[0..*] - кратність атрибута може приймати будь-яке позитивне ціле значення, що більше або дорівнює 0. Ця кратність може бути записана коротше у вигляді простого символу [*].

[1.:*] - кратність атрибута може приймати будь-яке позитивне ціле значення більше або рівне 1.

За замовченням приймає значення [1]

# Тип атрибутів

Тип атрибута являє собою вираз, семантика якого визначається мовою специфікації відповідної моделі. У найпростішому випадку тип атрибута вказується рядком тексту, що має осмислене значення в межах пакета або моделі, до яких належить розглянутий клас.

Наприклад:

: Color

: String

: Boolean

Тип атрибуту не є обов’язковим і може бути пропущена

# Значення за замовченням

Початкове значення слугує для завдання деякого початкового значення для відповідного атрибута в момент створення окремого екземпляра класу.

Тут необхідно дотримуватися правила належності значення типу конкретного атрибута.

Наприклад:

- color: Color = (255, 0, 0)
- name[1]: String = “Іван Іванович”
- visible: Boolean = True

Якщо початкове значення не вказано, то значення відповідного атрибута не визначено на момент створення нового екземпляра класу.

З іншого боку, конструктор відповідного об'єкта може перевизначати вихідне значення в процесі виконання програми, якщо в цьому виникає необхідність.

# Операції класів

# Операції

Операція описує поведінку класу і записується у другому блоці класу.

Повна форма виглядає так:

```
видимість назва (список параметрів): тип {властивості}
```

Наприклад:

```
+ checkBalanceOn (date: Date): Currency
```

# Список параметрів операції

Визначає перелік параметрів що передаються операції або повертаються

Повна форма виглядає так:

```
напрямок назва: тип = значення за замовченням
```

Наприклад:

```
in date: DateTime
inOut itemNumber: Integer
```

- `in` - вхідний параметр
- `out` - вихідний параметр
- `inOut` - вхідний та вихідний параметр

За замовченням `in`

# Видимість операції

Характеризує доступність цього операції з інших класів системи

\+ - **відкритий** (public) – така операція доступна з будь-якого іншого класу, пакета або модуля

\- - **закритий** (private) – така операція доступна тільки з цього класу

\# - **захищений** (protected) – така операція доступна тільки з підкласів цього суперкласу (при використанні спадкування)

Видимість не є обов’язковою і може бути пропущена

# Операції за типами

Слід розрізняти операції, що змінюють стан системи та операції, що не роблять цього.

**Запит** – є операцією, яка не змінює загальний стан системи а лише повертає значення якоїсь властивості. Такі операції можна позначати властивістю {query}

**Модифікатор** – це операція, яка змінює стан системи.

Інший варіант операцій:

**Отримання значення** - повертає значення якоїсь властивості

**Встановлення значення** – встановлює значення якоїсь властивості

# Операції та Методи

Часто операції та методи вважають одним і тим самим, проте слід вміти їх розрізняти.

**Операція** – представляє собою те, що викликається об’єктом під час створення процедури

**Метод** – тіло процедури

Їх має сенс розлічати в межах терміну поліморфізм.

Якщо є суперклас з якоюсь поведінкою та 2 підкласи, що перевизначають цю поведінку, то це одна операція та два методи, що її реалізують.

# Приклади операцій

`+створити()`

може позначати абстрактну операцію зі створення окремого об'єкта класу, яка є загальнодоступною і не містить формальних параметрів. Ця операція не повертає жодного значення після свого виконання.

`+намалювати(форма: Багатокутник = прямокутник, колір_заливки: Color = (0, 0, 255))`

може позначати операцію по зображенню на екрані монітора прямокутної області синього кольору, якщо не вказуються інші значення як аргументи даної операції.

`запросити_рахунок_клієнта(номер_рахунку:integer = 123456):Currency`

позначає операцію зі встановлення наявності коштів на поточному рахунку клієнта банку. При цьому аргументом цієї операції є номер рахунку клієнта, який записується у вигляді цілого числа (наприклад, "123456"). Результатом виконання цієї операції є деяке число, записане в прийнятому грошовому форматі (наприклад, $1,500.00).

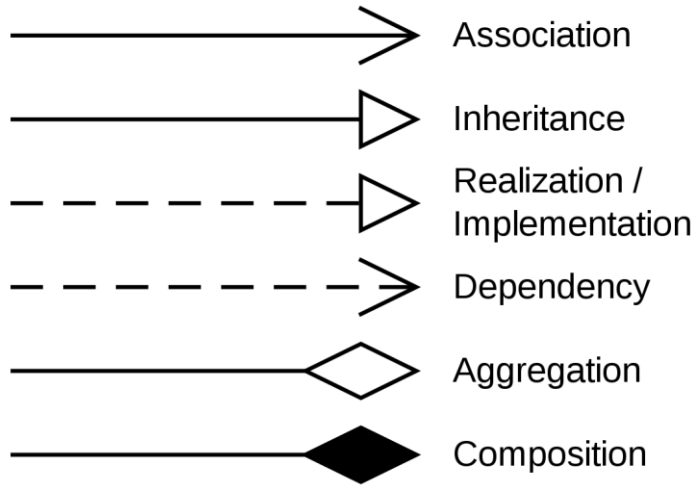
`видати_повідомлення():{"Помилка ділення на нуль"}`

сенс цієї операції не потребує пояснення, оскільки воно міститься в рядку-властивості операції. Це повідомлення може з'явитися на екрані монітора в разі спроби ділення числа на нуль, що неприпустимо.

# Відношення між класами

# Відношення між класами

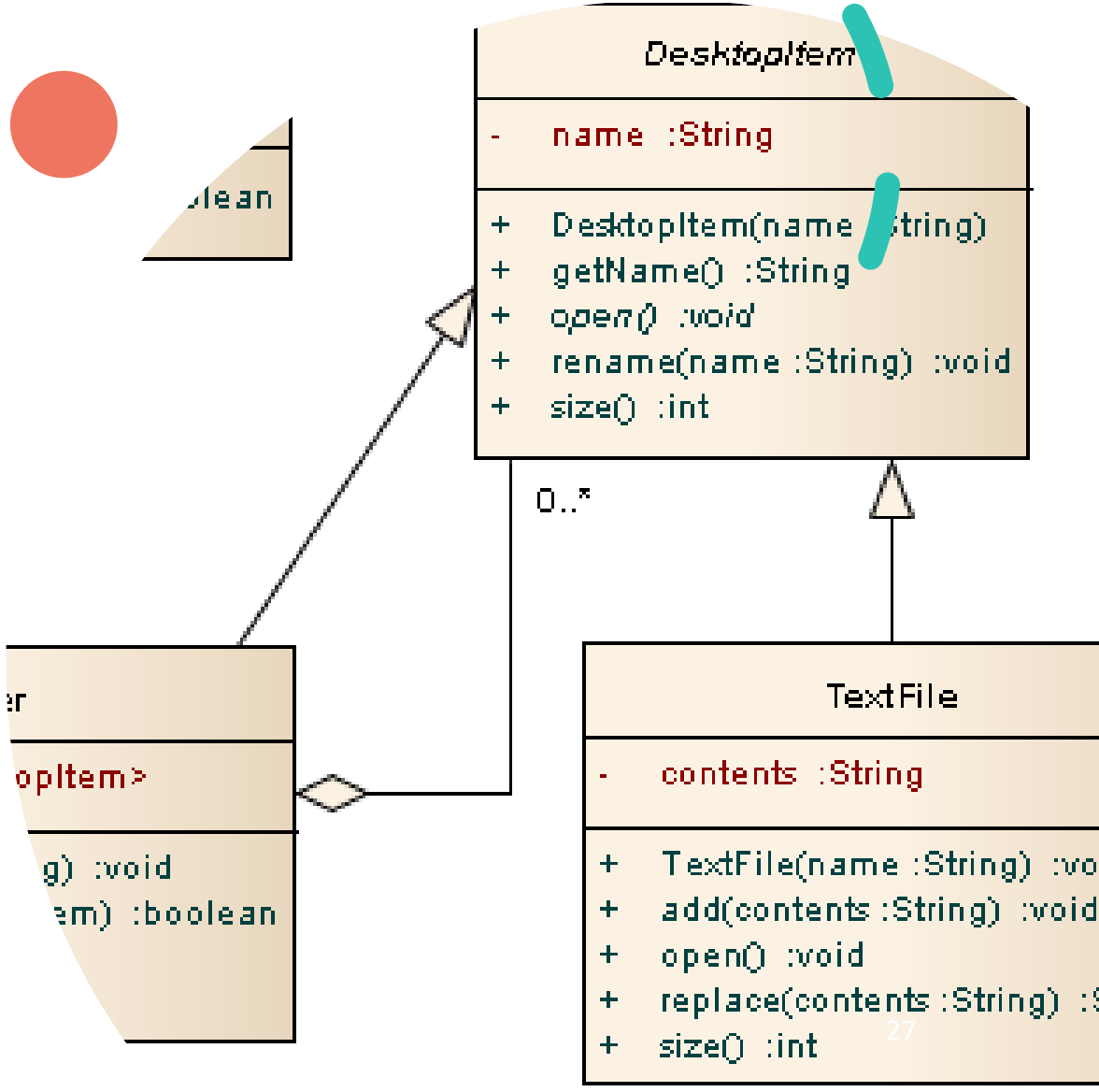
- **Відношення асоціації** (association)
- **Відношення залежності** (dependency)
- **Відношення агрегації** (aggregation)
- **Відношення композиції** (composition)
- **Відношення узагальнення** (generalization)
  або
  **Відношення спадкування** (inheritance)
- **Відношення реалізації** (realization)



# Відношення між класами

Відношення між класами показує що один клас звертається до іншого:

- Викликає метод
- Встановлює властивість
- Читає властивість
- Спадкує
- Реалізує функціональність
- тощо



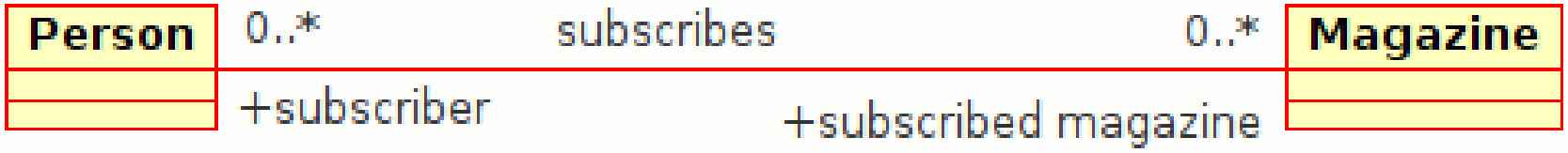
# Відношення асоціації

Показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності.

Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

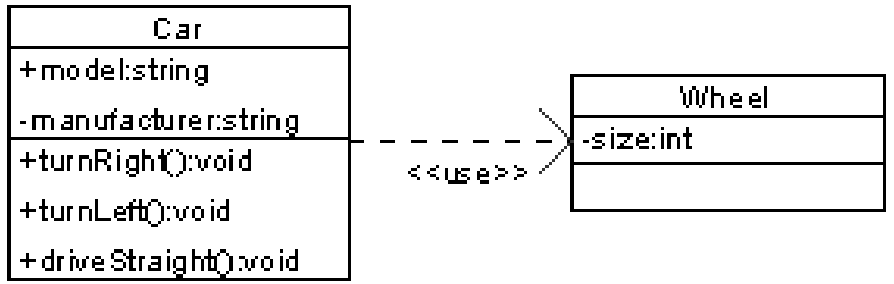
Асоціації може бути присвоєно ім'я, яке описує природу відносин.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity)



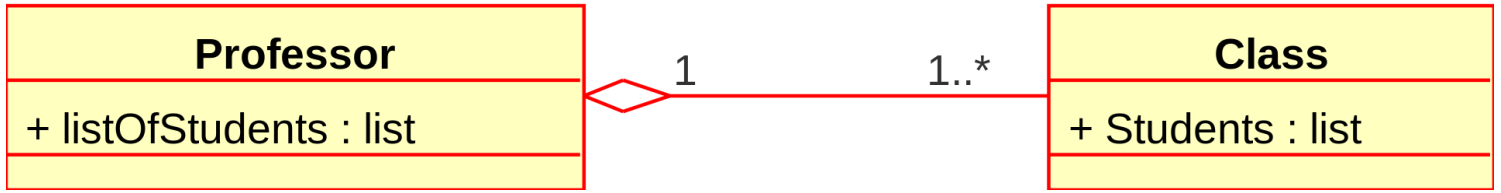
# Відношення залежності

- Відношення залежності використовується в такій ситуації, коли деяка зміна одного елемента моделі може вимагати зміни іншого залежного від нього елемента моделі.
- Позначається у вигляді стрілки від головного до залежного класу
- Це позначає, що якщо клас Car змінить свою поведінку або властивості (реалізацію), то клас Wheel теж має змінитись.



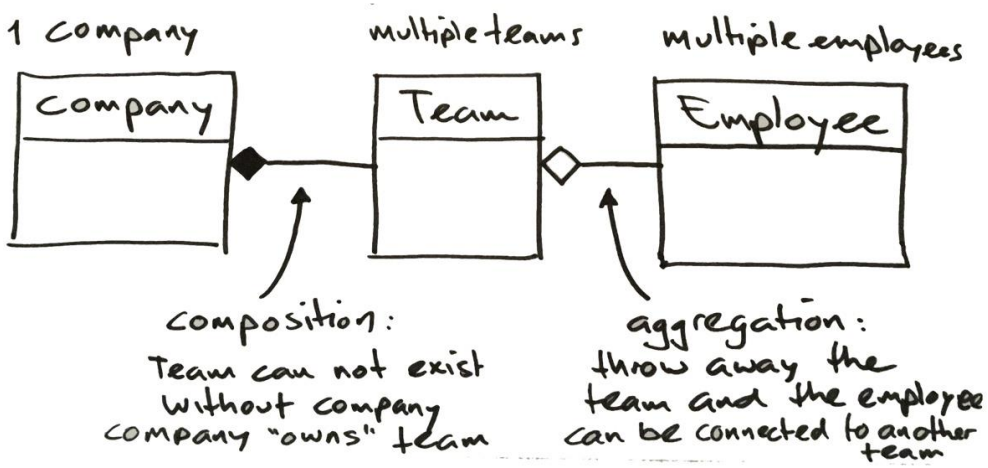
# Відношення агрегації

- Відношення агрегації має місце між кількома класами в тому разі, якщо один із класів являє собою деяку сутність, що включає в себе як складові частини інші сутності.
- За потреби можна вказувати кратність агрегації.
- Агрегація не накладає жорстких умов на термін існування об'єктів. Наприклад, «частини» можуть існувати тоді, коли «ціле» зникає.
- Графічно агрегація представлена порожнім ромбом на блоці «цілого», і лінією, яка проведена від цього ромба до класу, що міститься в ньому («частин»).



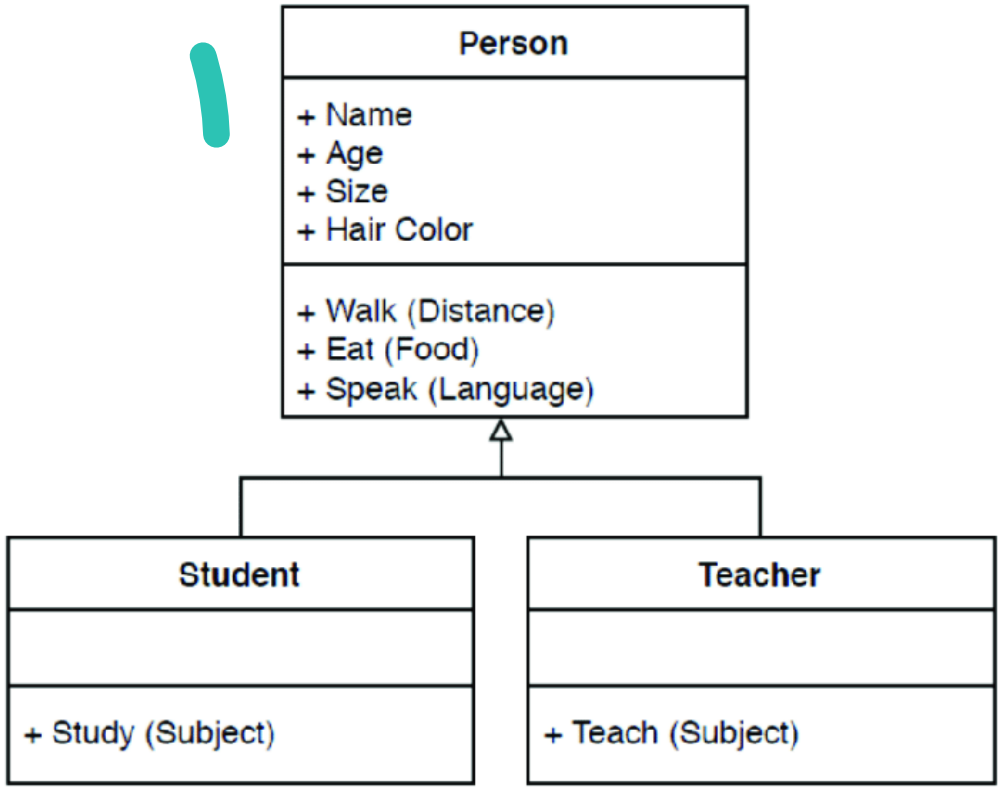
# Відношення композиції

- Більш строгий варіант агрегації
- Композиція має жорстку залежність часу існування екземплярів класу контейнера та екземплярів класів що містяться в ньому. Якщо контейнер буде знищений, то весь його вміст буде також знищено.
- Графічно представляється як і агрегація, але з зафарбованим ромбиком.



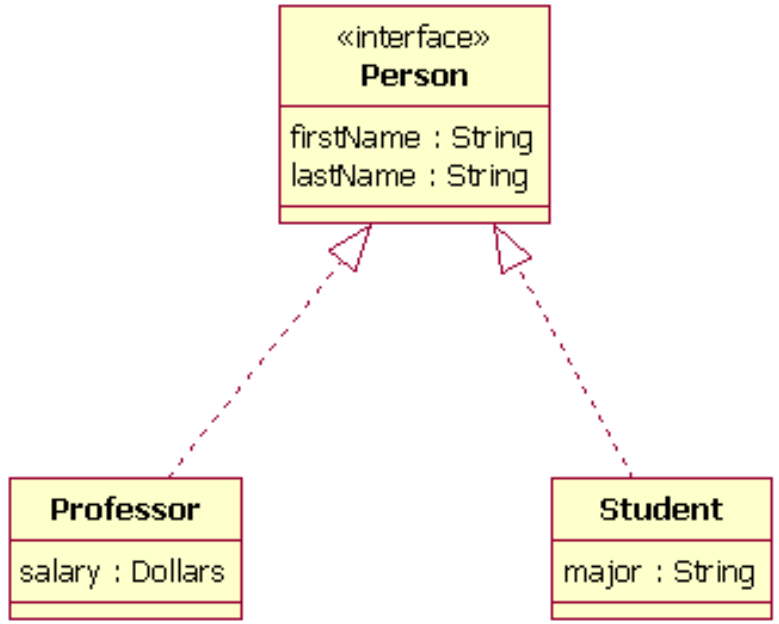
# Відношення узагальнення (спадкування)

- Це означає, що один з двох споріднених класів (підклас, підтип, нащадок) вважається спеціалізованою формою іншого (суперклас, супертип, батько), а суперклас вважається узагальненням підкласу.
- Графічне представлення узагальнення - це порожнистий трикутник на кінці лінії (або дерева ліній) суперкласу, який з'єднує його з одним або декількома підкласами.
- Відношення узагальнення також відоме як спадкування (наслідування) або відношення «є».



# Відношення реалізації

- зв'язок між двома елементами моделі, в якому один елемент моделі (клієнт) реалізує (впроваджує або виконує) поведінку, яку визначає інший елемент моделі (постачальник).
- Графічне представлення - порожнистий трикутник на «інтерфейсному» кінці пунктирної лінії
- Реалізації можуть бути показані тільки на діаграмах класів або компонентів.
- Зв'язок реалізації між класами/компонентами та інтерфейсами показує, що клас/компонент реалізує операції.



# Дякую

Ніколаєнко Дмитро
Володимирович