Лекція 8

Моделювання взаємодії

Ілюзія статичного коду

Коли ви дивитесь на код у GitHub або на діаграму класів, ви бачите "мертву" структуру без часу та навантаження. Але коли цей код запускається у Docker-контейнері під навантаженням 10k RPS, починається Runtime — динамічний аспект поведінки системи у часі.

Найпотужнішим інструментом для візуалізації часової послідовності подій є Sequence Diagram.

1

Код статичний, виконання динамічне

Більшість складних помилок живуть у динаміці, а не в структурі коду.

2

Об'єкт ≠ Клас

У динаміці ми завжди оперуємо конкретними екземплярами (user1, user2), які можуть конфліктувати між собою.

3

Моделюємо сценарії

Неможливо намалювати "діаграму роботи всього банку". Можна намалювати "діаграму переказу SWIFT".

Статика vs Динаміка

Щоб чітко розмежувати ці поняття, розглянемо ключові відмінності між статичним та динамічним моделюванням. Це те, що відрізняє підхід Junior-розробника від підходу архітектора.

Характеристика	Статичне моделювання	Динамічне моделювання
Основне питання	З чого це зроблено?	Як це працює?
Одиниця виміру	Клас, Інтерфейс, Таблиця	Об'єкт (Екземпляр), Потік
Фактор часу	Ігнорується (миттєво)	Критичний (послідовність, затримки)
Типові помилки	Порушення SOLID, циклічні залежності	Race Conditions, Deadlocks, Timeouts
Аналог у житті	Анатомічний атлас людини	Відеозапис бігу спортсмена
Інструмент діагностики	Code Review / Linter	Debugger / Distributed Tracing (Jaeger)

Проблема подвійного списання

Розглянемо класичну ситуацію динамічного моделювання, яку неможливо описати статичною діаграмою. Користувач натискає кнопку "Купити" двічі дуже швидко.

Статика

У нас є метод buy(), який перевіряє баланс і списує гроші. Код виглядає правильним.

Динаміка (Race Condition)

- Потік 1 читає баланс (100 грн)
- Потік 2 читає баланс (100 грн) — до того, як Потік 1 списав кошти
- Потік 1 списує 100 грн
- Потік 2 списує 100 грн
- Результат: Баланс -100 грн

Саме для виявлення таких сценаріїв ми будуємо діаграми взаємодії ДО написання коду.

Sequence Diagram: Кінострічка вашого коду

Якщо Use Case — це афіша фільму (про що кіно), а ERD — це список акторів, то Sequence Diagram — це власне сценарій. Це покадровий опис того, хто що говорить і в якому порядку.

Проєктування REST API

Коли ви проєктуєте REST API, ви малюєте Sequence Diagram для визначення потоку запитів.

Діагностика проблем

Коли розбираєтесь, чому транзакція зависає, ви малюєте Sequence Diagram для аналізу.

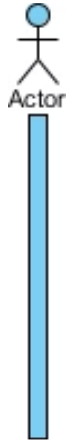
Архітектурне проєктування

Це діаграма №1 для бекенд-розробників та архітекторів в індустрії.

> **Головна мета:** Показати взаємодію об'єктів у суворій часовій послідовності.

Система координат: Час і Простір

Діаграма будується у двовимірному просторі, де кожна вісь має чітке значення. Розуміння цієї системи координат є фундаментальним для правильного читання та створення діаграм послідовності.

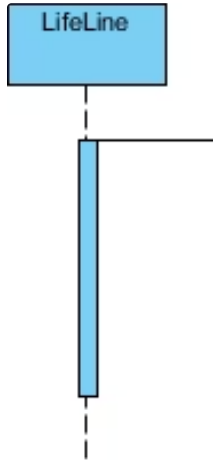


Учасник (Actor)

Суцільна лінія з трикутником. Один об'єкт просить інший щось зробити.

Лінія життя (Lifeline)

Голова: Прямокутник із назвою.
Хвіст: Пунктирна лінія, що тягнеться вниз.

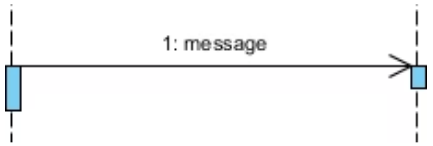


Фокус управління (Activation Bar)

Тонкий довгий прямокутник, накладений на пунктирну лінію.

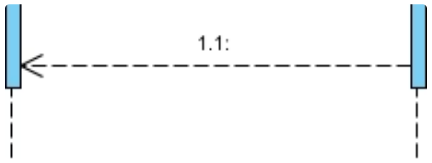
Види повідомлень: Синтаксис стрілок

Стрілочками позначаються виклики одними елементами інших та дані, які повертаються в результаті викликів.



Виклик (Call)

Суцільна лінія з трикутником. Один об'єкт просить інший щось зробити.

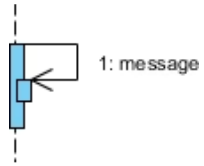


Відповідь (Return)

Пунктирна лінія. Повернення результату. Малюйте тільки для важливих даних.

Рефлексія (Self Message)

Стрілка виходить з активації і повертається в ту саму активацію.

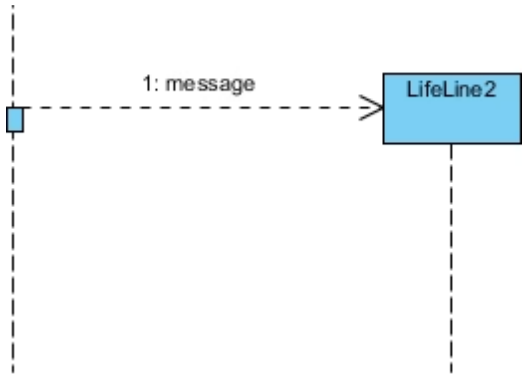


Рекурсія (Recursive Message)

Стрілка виходить з активації і повертається в ту саму активацію рекурсивно.

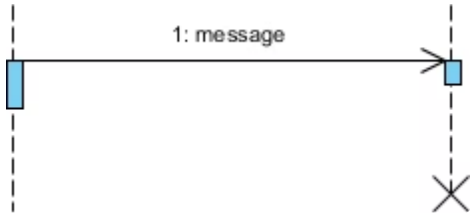
Життєвий цикл об'єктів

Більшість об'єктів на діаграмі малюються у самому верху - вони вже існували до початку сценарію. Але якщо логіка вимагає створення нового екземпляра, використовуємо специфічну нотацію.



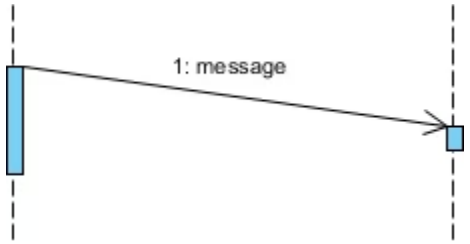
Створення об'єкта (Create Message)

Стрілка вказує прямо на прямокутник (голову) об'єкта, а не на його лінію життя.



Знищення об'єкта (Destroy Message)

Лінія життя закінчується великим хрестиком (Stop Point).



Тривале повідомлення (Duration Message)

Символ об'єкта малюється нижче та показує відстань між двома моментами часу

Все, що має початок, має і кінець. Коли об'єкт виконав свою роль, він має звільнити ресурси

Типові сценарії життєвого циклу

В архітектурі виділяємо три патерни життя об'єктів на діаграмах. Розуміння цих патернів допомагає правильно проєктувати систему та оптимізувати використання ресурсів.

- **Довгожителі (Long-Lived)**

  Хто: Сервіси, Репозиторії, Синглтони

  Вигляд: Прямокутник на самому верху, лінія життя йде до низу сторінки

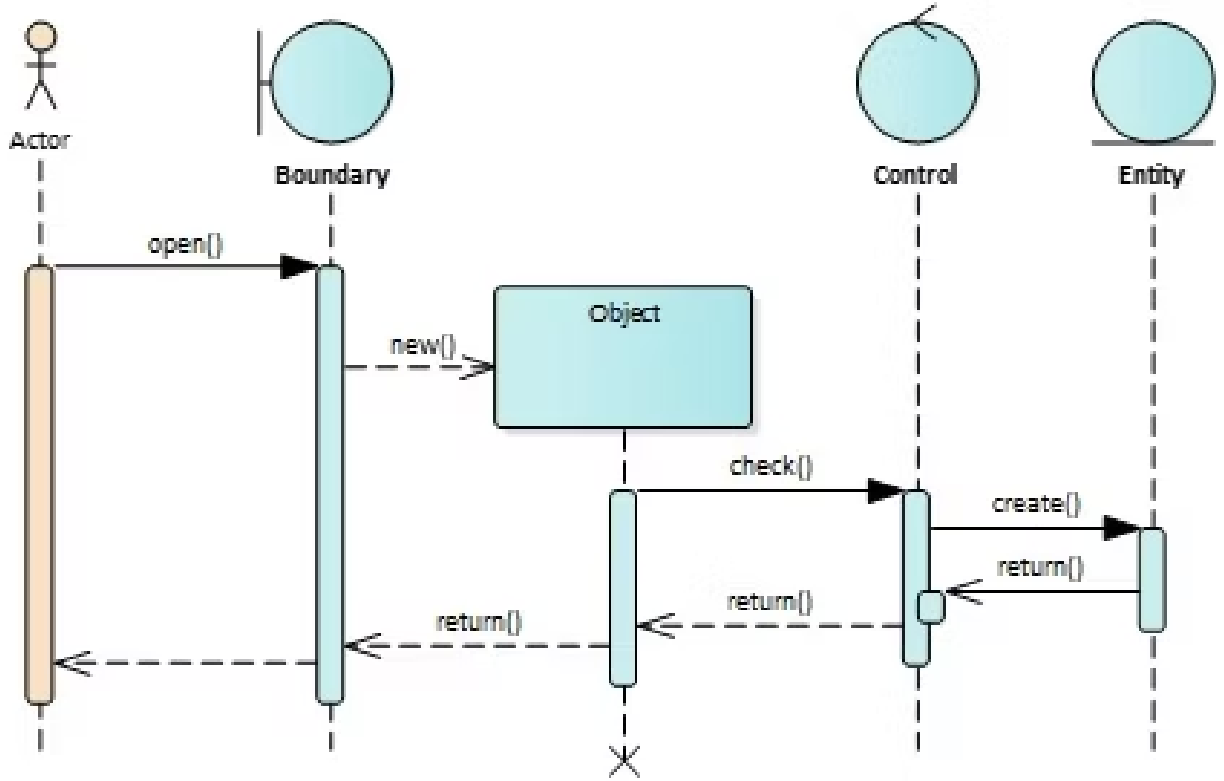
  Суть: Складають "скелет" додатку

- **Тимчасові об'єкти (Transient)**

  Хто: DTO, Події (Events), Виключення (Exceptions)

  Вигляд: Створюються посеред діаграми, живуть кілька кроків, знищуються (X)

  Приклад: HttpRequest створюється, передається через ланцюжок і вмирає після відправки HttpResponse



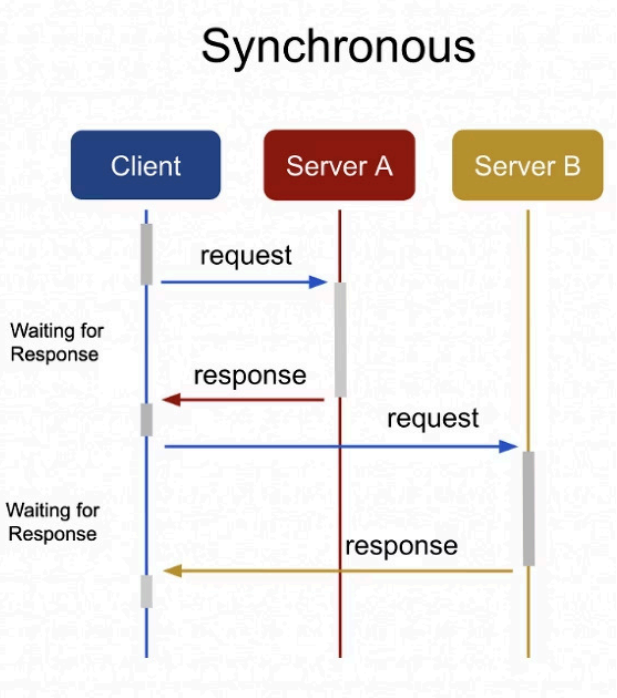
Синхронні та асинхронні повідомлення

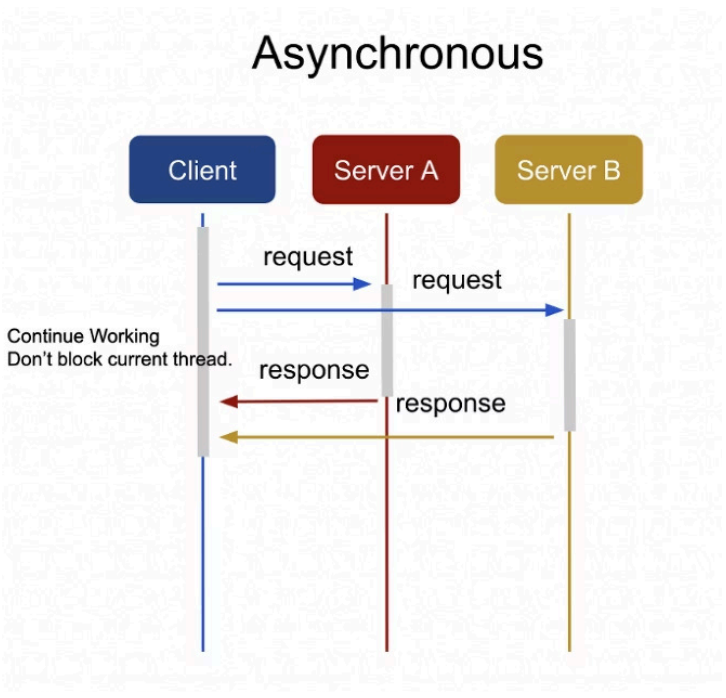
Синхронне повідомлення (Synchronous Call)

Це стандартний режим роботи більшості функцій та HTTP-запитів. Sender передає управління Receiver і блокується — чекає, поки не отримає return.

Асинхронне повідомлення (Asynchronous Call)

Це основа Event-Driven Architecture та мікросервісів. Sender відправляє сигнал і миттєво переходить до виконання наступного рядка коду. Він не чекає результату обробки.





Практичний кейс: Замовлення кави

Порівняємо два підходи на прикладі кав'ярні, щоб побачити різницю між синхронним та асинхронним виконанням.

1 Сценарій А: Синхронний (Bad UX)

1. Клієнт → Касир: Замовити Каву
2. Касир → Бариста: Варити()
3. Клієнт і Касир стоять і дивляться 5 хвилин
4. Бариста → Касир: Кава готова
5. Касир → Клієнт: Ваша кава

Проблема: Черга не рухається

2 Сценарій Б: Асинхронний (Good UX)

1. Клієнт → Касир: Замовити Каву
2. Касир → Черга: Додати тікет (асинхронно!)
3. Касир → Клієнт: Ваш номер 42, чекайте
4. Клієнт відходить. Касир обслуговує наступного
5. Бариста (пізніше) → Табло: 42 готове
6. Клієнт → Бариста: Забрати

Стрілка від Касира до Черги відкрита (□), лінія життя Касира одразу звільняється

Висновки про синхронність та асинхронність

1 Стрілка має значення

Зафарбований трикутник = чекаємо. Відкритий наконечник = не чекаємо.

2 Синхронні виклики для читання

Використовуйте для операцій читання (GET), де користувачу треба дані "тут і зараз".

3 Асинхронні виклики для важких операцій

Використовуйте для важких операцій запису (POST створення звіту, надсилання листів), щоб не блокувати інтерфейс.

4 Виявлення вузьких місць

Sequence Diagram — найкращий спосіб побачити "пляшкові шийки" (Bottlenecks) синхронної архітектури ще до написання коду.

Алгоритми не бувають ідеально прямими. Що, якщо платіж не пройшов? Що, якщо товарів у кошику п'ять? Для цього UML має спеціальні блоки — Фрагменти взаємодії.

Анатомія взаємодії

Будь-який фрагмент виглядає як прямокутна рамка навколо частини діаграми. Це потужний інструмент для опису складної логіки у Sequence Diagram.

1 Оператор (Operator)

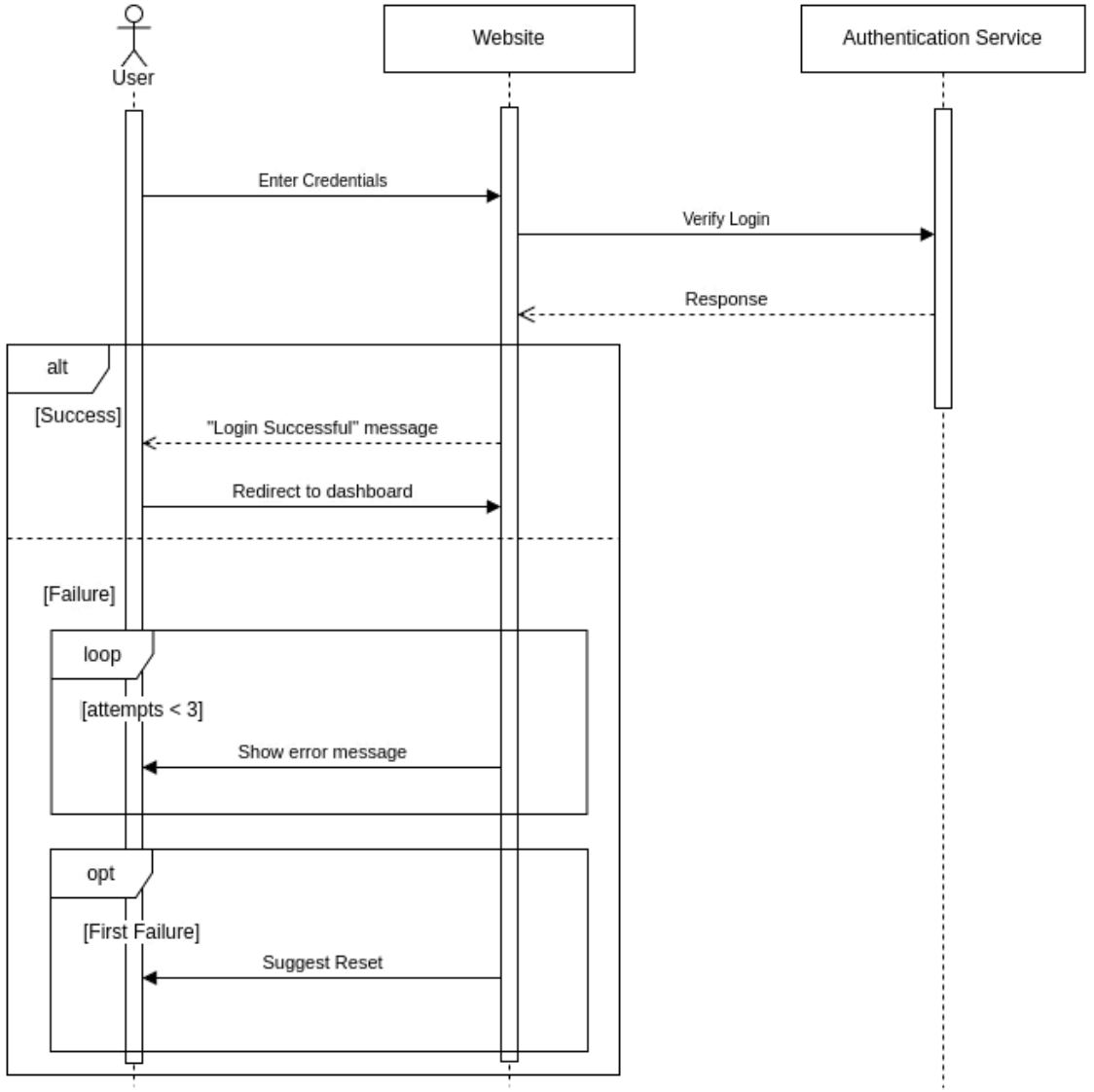
У лівому верхньому кутку в п'ятикутнику написано ключове слово (alt, opt, loop, par). Воно визначає тип логіки.

2 Операнди (Operands)

Внутрішні секції рамки, розділені горизонтальною пунктирною лінією для різних варіантів виконання.

3 Вартова умова (Guard Condition)

Логічний вираз у квадратних дужках [x > 0], розміщений над лінією життя. Якщо він true, вміст виконується.



Просунуті оператори: Concurrency & Interrupts

Це рівень Senior/Architect. Ці фрагменти описують складну поведінку розподілених систем та критичні сценарії обробки помилок.

Паралельність (par)

Всі секції стартують одночасно. Порядок виконання не гарантований.

Приклад: Реєстрація — паралельно: зберегти в SQL, відправити Email, відправити подію в Kafka

Переривання (break)

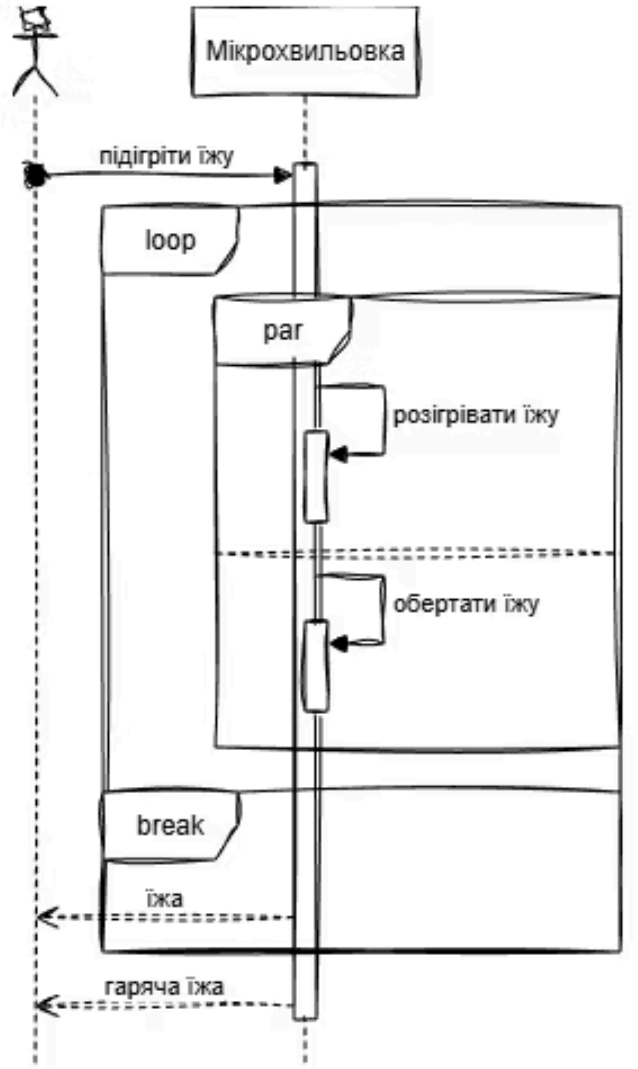
"Червона кнопка" сценарію. Якщо умова виконується — весь сценарій негайно припиняється.

Приклад: break [Token Invalid]: Повернути 401. До бази ніхто не йде

Посилання (ref)

Принцип DRY в діаграмах. "Гіперпосилання" на іншу діаграму.

Приклад: Замість малювання 20 стрілок "Авторизація" — прямокутник ref "Login Sequence"



Типові помилки та Best Practices

Вкладеність (Inception)

Помилка: alt всередині loop всередині alt. Більше 2-х рівнів вкладеності — діаграма нечитабельна.

Рішення: Винесіть внутрішню логіку в окрему діаграму і використайте ref.

Код у діаграмі

Помилка: Складний код у вартових умовах [user.getAge() > 18 && !user.isBanned()].

Рішення: Пишіть бізнес-мовою: [User is valid].

Зловживання par

Помилка: Використання par, коли порядок дій важливий.

Рішення: Паралельність означає хаос у часі. Використовуйте тільки для справді незалежних операцій.

Висновки до Лекції 8

Ми завершили розгляд моделювання взаємодії. Тепер ви знаєте, як показати "кіно" про роботу вашої системи та проєктувати складні алгоритми до написання коду.

Головний інструмент

Sequence Diagram — ваш головний інструмент для дизайну алгоритмів

Lifeline та Activation

Показують, хто працює, а хто чекає у системі

Async — ключ до продуктивності

Відкрита стрілка — основа високопродуктивних систем

Fragments для реальної логіки

alt, loop, break дозволяють описати реальну логіку, а не ідеальний світ

> **Що далі?** Ми навчилися моделювати поведінку групи об'єктів. Але іноді складність ховається всередині одного об'єкта. Наприклад, "Замовлення" може бути: Нове → Оплачене → В обробці → Доставлене → Скасоване. Як описати ці переходи? Для цього існує Діаграма станів (State Machine Diagram) — тема наступної лекції.