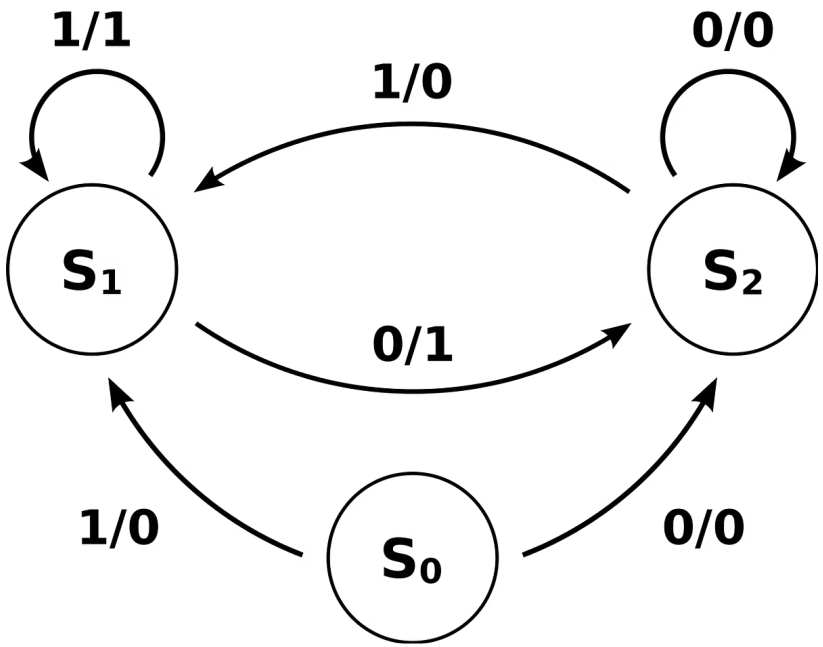
Лекція 9

Моделювання поведінки та станів



Проблема "Boolean Flags Hell"

Уявіть клас Order (Замовлення) з набором булевих полів: isPaid, isShipped, isCancelled, isDelivered. Математично це дає нам $2^4 = 16$ можливих комбінацій.

Питання: Чи має сенс комбінація isShipped = true AND isCancelled = true? У реальному житті — ні. Але в коді це можливо, якщо програміст забув скинути прапорець. Це джерело багів, коли система потрапляє в невалідний стан.

State Machine (Кінцевий автомат) — це математична модель, яка забороняє об'єкту перебувати в нелогічних станах. Замість 16 комбінацій прапорців ми визначаємо 4 чітких стани і жорсткі правила переходу між ними.

Анатомія Автомата

Стан (State)

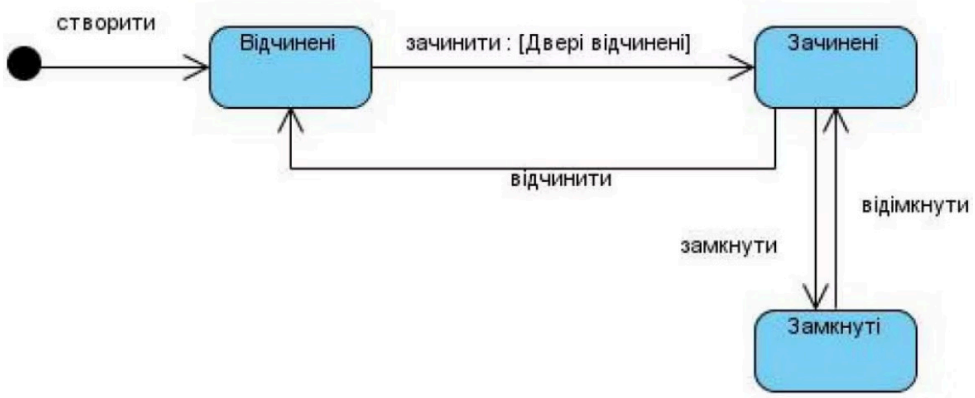
Умова або ситуація в житті об'єкта, протягом якої він задовольняє певну умову, виконує діяльність або очікує на подію. Має тривалість і стабільність.

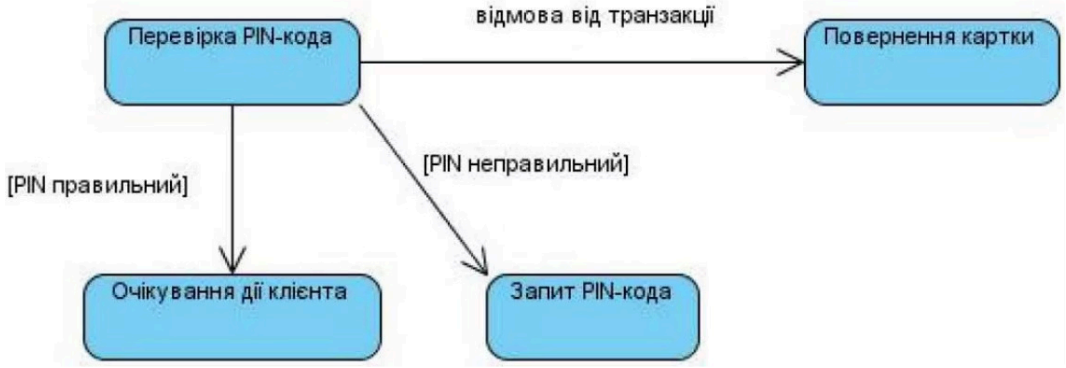
Подія (Event)

Стимул, який може спровокувати зміну стану. Типи: Call Event (виклик методу), Signal Event (асинхронне повідомлення), Time Event (спливання часу), Change Event (зміна даних).

Перехід (Transition)

Атомарний шлях з одного стану в інший. Формула: Source State ⊠ Event [Guard] / Action ⊠ Target State. Об'єкт не може застрягти між станами.





Практичний приклад: Турнікет метро

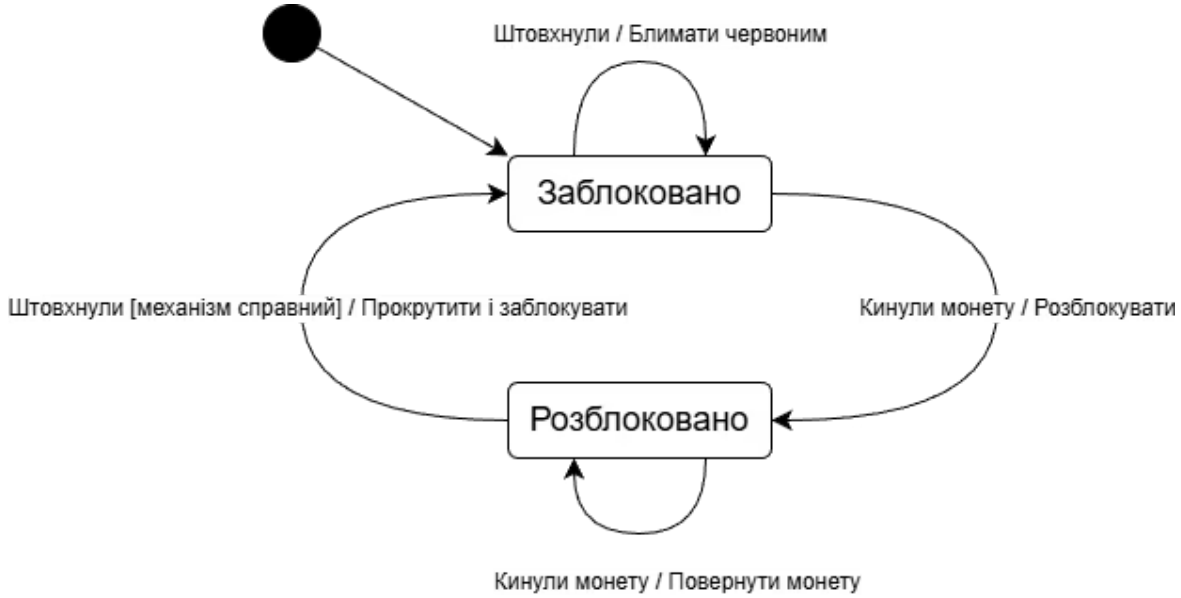
Класичний FSM

Стани: Locked (Заблоковано), Unlocked (Розблоковано)

Події: Coin (Кинули монету), Push (Штовхнули)

Логіка переходів

1. Locked + Coin → Unlocked (Дія: Розблокувати механізм)
2. Unlocked + Push → Locked (Дія: Прокрутити і заблокувати)
3. Locked + Push → Locked (Перехід у себе. Дія: Блимати червоним)
4. Unlocked + Coin → Unlocked (Перехід у себе. Дія: Повернути монету)



Ця модель робить неможливим прохід без оплати на рівні архітектури.

Коли проста схема не працює

1 Стан — це якісна характеристика

Стан об'єкта — це не просто значення змінних, а якісна характеристика його поведінки.

2 Заміна складної логіки

State Machine замінює складну логіку if-else на чіткий граф переходів, що робить код зрозумілішим.

3 Структура переходу

Перехід складається з Події, Вартової умови та Дії — три компоненти для повного контролю.

4 Надійне моделювання

Це єдиний надійний спосіб моделювати життєвий цикл складних бізнес-сутностей: Замовлення, Документ, Транзакція.

Але що робити в складніших випадках?

- **50 станів**

  Діаграма перетвориться на павутиння, яке неможливо читати

- **Паралельні стани**

  Об'єкт може бути в кількох станах одночасно (персонаж "Біжить" і "Перезаряджає зброю")

- **Пам'ять стану**

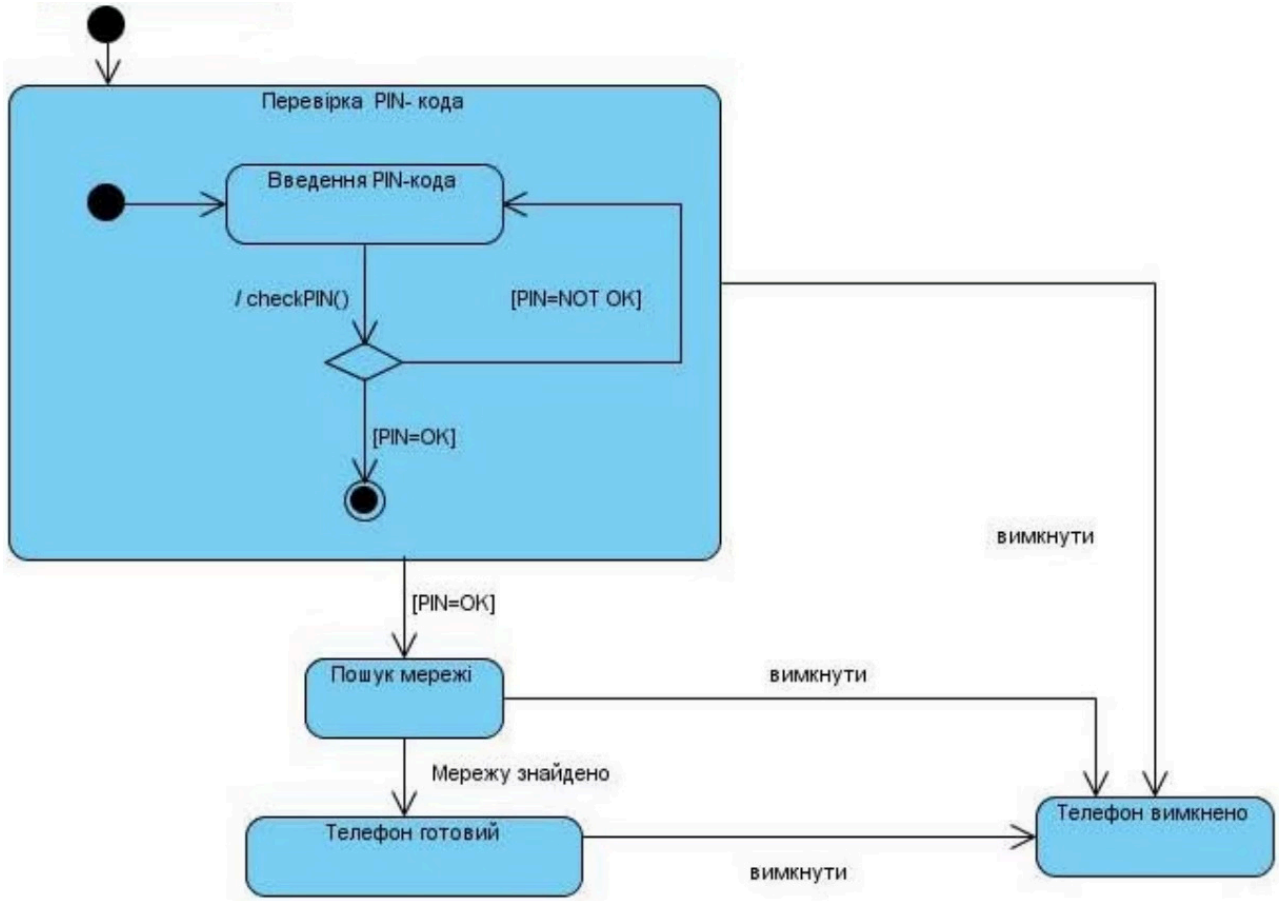
  Об'єкт повинен "пам'ятати", що він робив до переривання (музика грає з того ж місця після паузи)

Для цього UML State Machine Diagram має розширений синтаксис — найпотужнішу діаграму для системних архітекторів.

Боротьба зі складністю: Складені стани

Композитний стан (Composite State)

Замість малювання стрілки вимкнути від кожного окремого стану, ми обгортаємо їх у великий стан і малюємо одну стрілку вимкнути від зовнішнього кордону. Це зменшує кількість стрілок у рази.



Історичні стани (History State)

Що відбувається, коли ви виходите з композитного стану, а потім повертаєтесь?

- **За замовчуванням:** Ви починаєте спочатку (з Initial node підстану)
- **З історією:** Ви повертаєтесь туди, де зупинилися

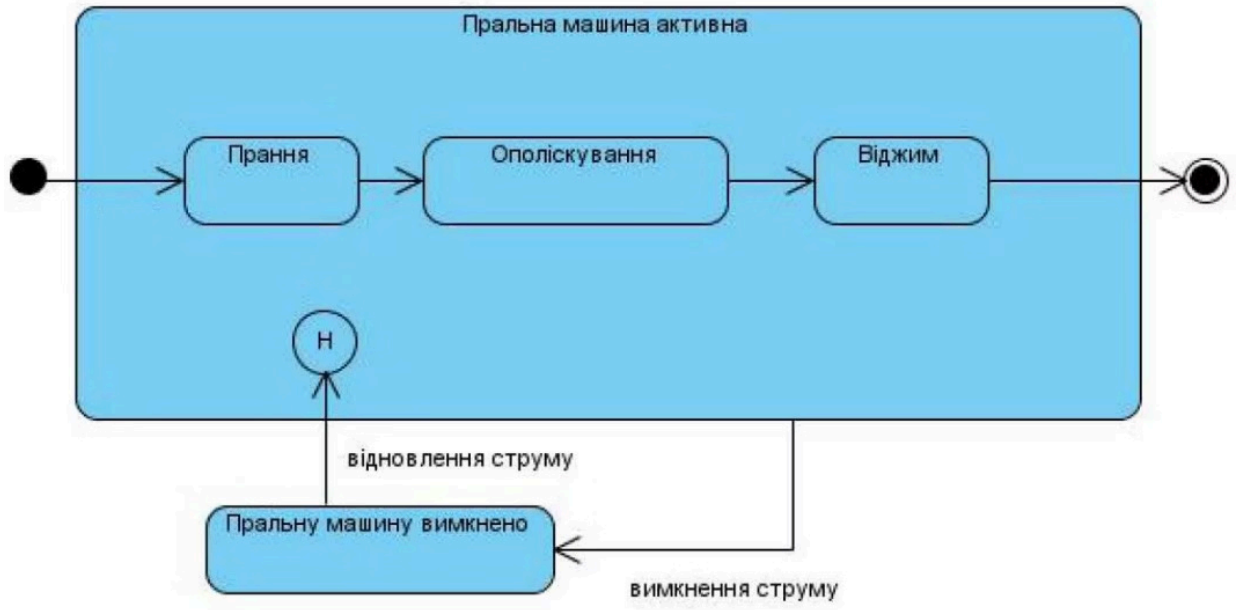
Shallow History (H)

Пам'ятає тільки останній активний підстан на одному рівні вкладеності

Deep History (H*)

Пам'ятає весь ланцюжок вкладеності (якщо у вас були підстани всередині підстанів)

Приклад: Ви слухали музику (стан Playing), натиснули "Відповісти на дзвінок" (перехід у Call), а потім завершили дзвінок. Перехід веде до (H) всередині музичного плеєра, тому система відновлює стан Playing, а не скидає в Stopped.



Стан — це не просто наліпка

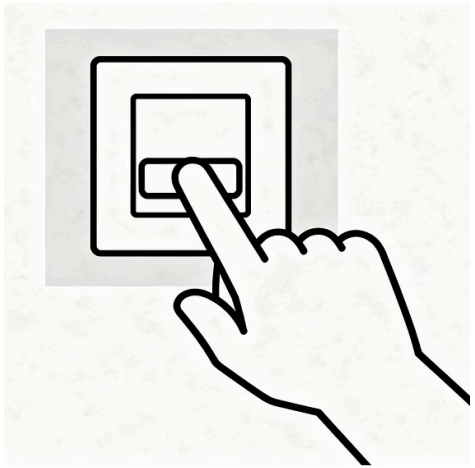
Досі ми розглядали стани як пасивні мітки: "Замовлення оплачене". Але в реальному програмуванні стан — це активний контейнер. Коли об'єкт потрапляє в стан, він не просто "сидить", він виконує роботу.

Вхід у кімнату

Ви вмикаєте світло — це **Entry action**

Перебування в кімнаті

Ви сидите і читаєте книгу — це **Do activity**



Вихід з кімнати

Ви вимикаєте світло перед виходом — це **Exit action**

UML дозволяє чітко розмежувати ці три типи активності, що критично важливо для управління ресурсами та уникнення витоків пам'яті.

Анатомія внутрішньої секції стану

Якщо "розкрити" прямокутник стану, ми побачимо внутрішній відсік, де прописуються дії.

Вхідна дія (entry)

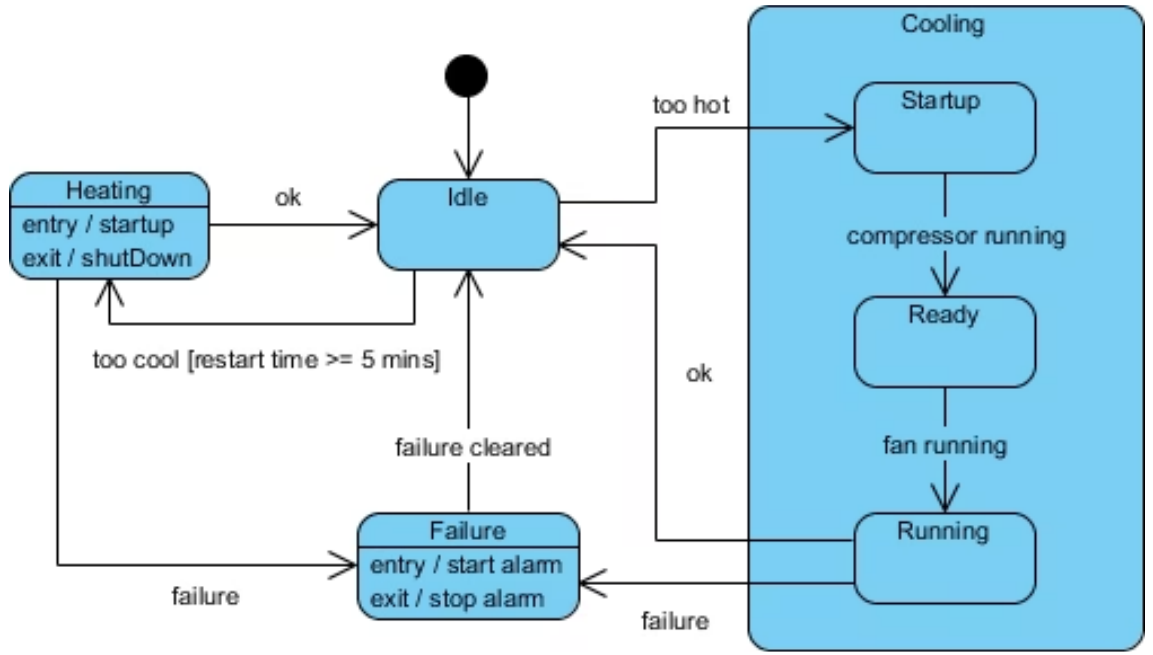
Атомарна, миттєва, блокуюча.
Виконується одразу при вході в стан.

Діяльність (do)

Тривала, переривчаста, виконується в окремому потоці. Починається після entry і триває доти, доки діяльність завершиться або прийде подія виходу.

Вихідна дія (exit)

Атомарна, миттєва, блокуюча.
Виконується перед виходом зі стану.



Статус як хребет бізнес-логіки

У будь-якій корпоративній системі (CRM, ERP, Banking) центральним елементом є Документ (Замовлення, Рахунок, Договір) або Транзакція (Платіж, Переказ). Уся бізнес-логіка крутиться навколо одного поля в базі даних — **status**.

status == DRAFT

Користувач може редагувати всі поля

status == APPROVED

Редагувати заборонено, тільки скасувати

status == ARCHIVED

Документ доступний тільки для читання

Проблема: Початківці часто хардкодять ці правила (if status == 'DRAFT'...) по всьому коду.

Рішення: Спроєктувати чітку State Machine, де переходи та права доступу визначені на рівні архітектури.

Патерн "Життєвий цикл Документа"

Більшість документів проходять через три мета-стани. Це універсальний шаблон.

Фаза 1: Чернетка (Mutable Phase)

Стани: Draft, New, Editing

- Документ мінливий (Mutable). Можна змінювати поля, додавати позиції
- Документ не має юридичної сили. Не впливає на звіти чи баланс
- Можна фізично видалити (DELETE)

Фаза 2: Активний (Immutable / Locked Phase)

Стани: Pending Approval, Approved, Signed, In Progress

Перехід: Зазвичай через подію Submit або Sign

- Документ стає незмінним (Immutable)
- Має юридичну силу (резервує товар, списує гроші)
- Фізичне видалення заборонено

Фаза 3: Термінальний (Final Phase)

Стани: Completed, Rejected, Cancelled, Archived

- Це "Чорна діра". Звідси немає виходу (окрім рідкісних кейсів Reopen)
- Документ зберігається для історії (Audit Log)

Патерн "Транзакція" (Transaction Lifecycle)

Транзакції (платежі) відрізняються від документів. Вони живуть швидко, автоматично і вимагають гарантії узгодженості (Consistency).

1. **Created / Initial**

   Запис створено в БД, але гроші ще не рухались

2. **Pending / Processing**

   Запит відправлено в банк/шлюз. Ми чекаємо (стан невизначеності). Не можна ініціювати повторну оплату (захист від подвійного списання)

3. **Authorized**

   Гроші заблоковано на карті клієнта (Hold)

4. **Settled / Captured**

   Гроші списано фінально (Термінальний стан успіху)

5. **Failed / Declined**

   Помилка (немає коштів, збій мережі). Термінальний стан помилки

6. **Refunded / Voided**

   Повернення коштів (Окремий термінальний стан)

Архітектурна порада: Ніколи не використовуйте один статус Failed для всього. Розділяйте: Hard_Fail (карта вкрадена — не пробувати знову) і Soft_Fail (тайм-аут мережі — можна спробувати ще раз через хвилину).

Висновки

Ми завершили великий блок моделювання поведінки. Тепер ви розумієте ключові принципи проєктування складних систем.

State Machine — захист від помилок

Це найкращий спосіб уникнути логічних помилок у життєвому циклі об'єктів

Стан — це процес, а не точка

Стан — це процес, а не точка.

entry/exit — гарантія ресурсів

Використовуйте entry/exit для гарантованої ініціалізації та звільнення ресурсів. Це ваш захист від витоків пам'яті.

do — для тривалих процесів

Використовуйте do для тривалих процесів, які мають перериватися при зміні стану.

Зовнішній vs внутрішній перехід

Розрізняйте зовнішній перехід (перезавантаження стану) і внутрішній перехід (реакція без зміни стану).

Що далі? Остання лекція буде присвячена діаграмі класів та патернам проєктування — готовим рецептам вирішення типових проблем архітектури.