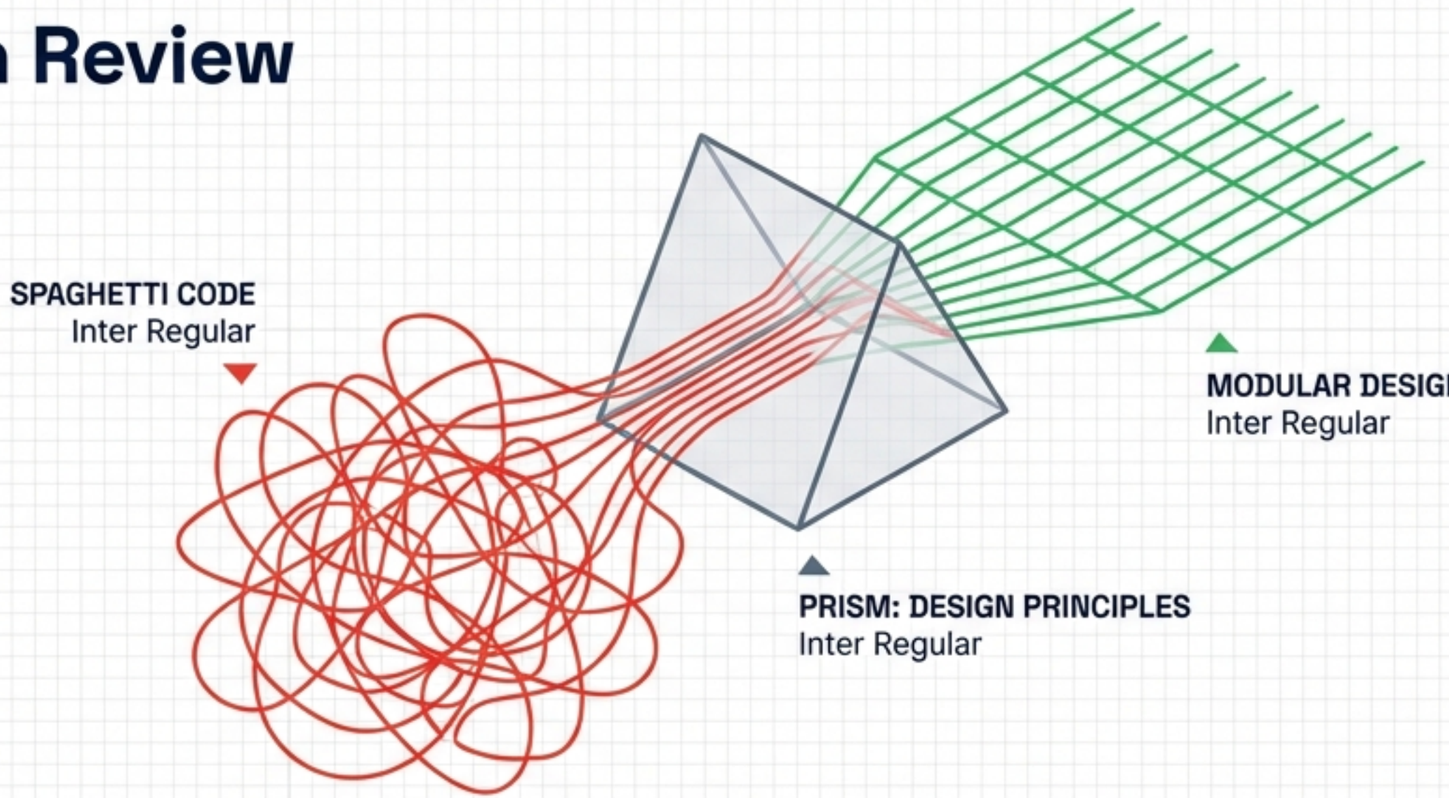
SWE 4743 Exam 1: Design as Warfare Against Complexity

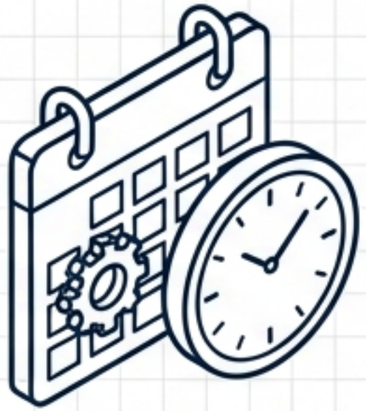
Object-Oriented Design Review



- **THE MISSION:** Transform from a tactical coder to a strategic designer.
- **THE SCOPE:** Lectures 1–6 (LSP), Ousterhout Ch 1–8, Assignments 1 & 2.
- **THE GOAL:** Make change safe and affordable.

Working Code is Not Enough.

The Assessment Landscape

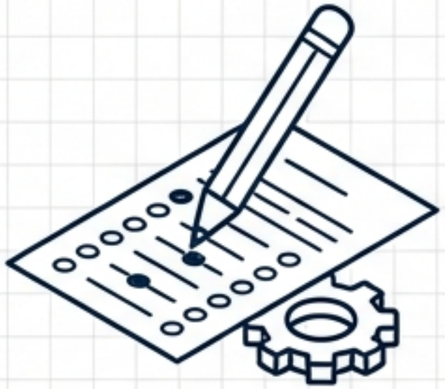


Logistics

Format: In-person, written.

Constraints: Closed-book, no devices.

Duration: Standard class period.

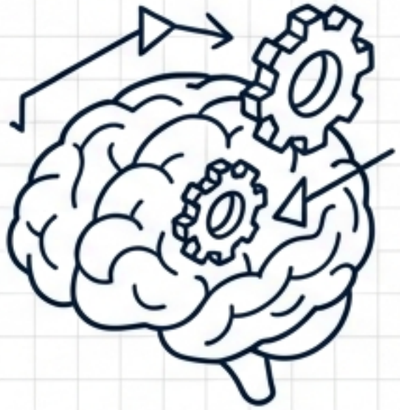


Question Types

True/False: Conceptual checks.

Multiple Choice: Diagnosis.

Short Answer: Design correction.



Expectations

Bloom Levels 3 & 4 (Analysis & Application).

Not just definitions—diagnosis required.

Prerequisites: Polymorphism, Encapsulation, Inheritance, Big O.

The Enemy is Complexity

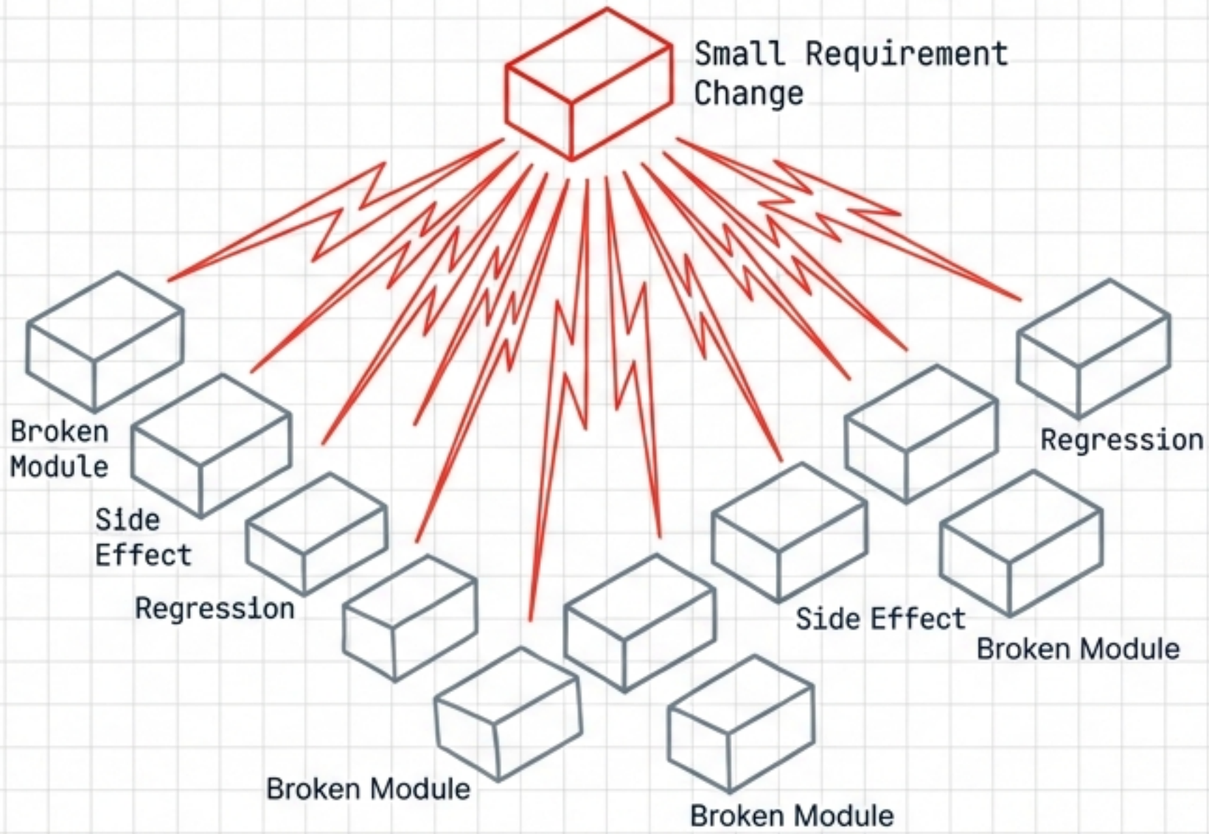
Definition: Complexity is defined by Change Amplification and Cognitive Load.

Source: Ousterhout, Chapters 1 & 2.

Insight: Complexity accumulates in increments. “Just one more if-statement” is the root of system rot.

The Golden Thread: We design to lower the cost of the future.

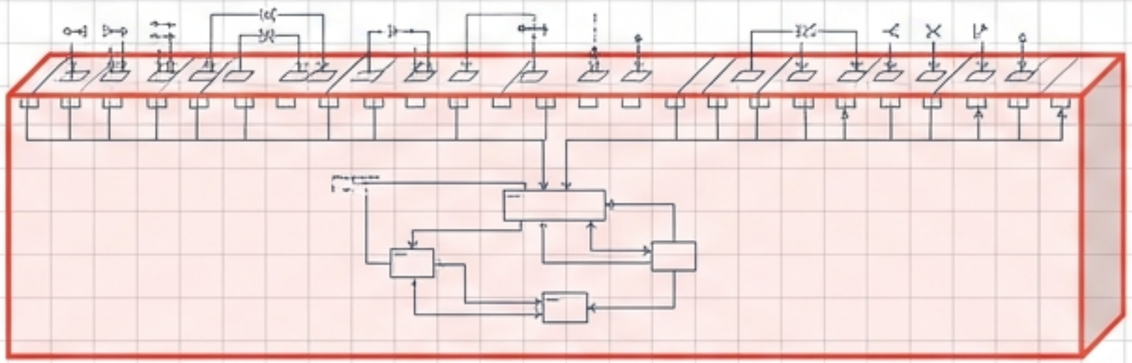
Change Amplification



Tactical vs. Strategic Programming

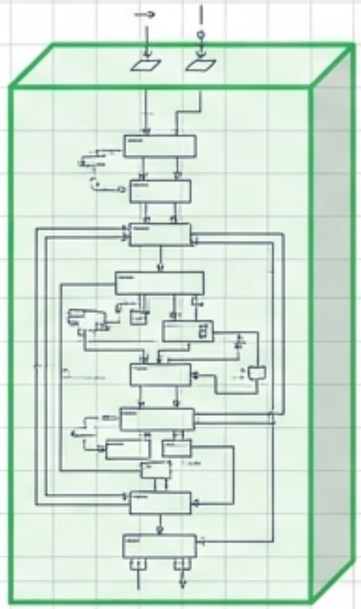
Modules Should Be Deep (Ousterhout Ch 4)

Shallow Module



Tactical: Works now, complex later.
Exposed implementation.

Deep Module



Strategic: Simple interface, powerful functionality hidden below.

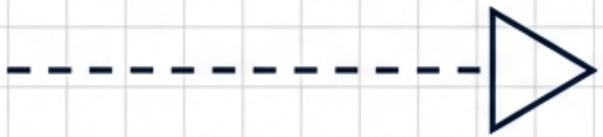
Pull Complexity Downwards: Keep ‘Main’ simple. Push hard logic deep.

Mapping the Structure with UML

UML maps Static Structure, not runtime flow. It reveals coupling.

Inheritance (Generalization)

IS-A relationship. Structural & Compile-time.

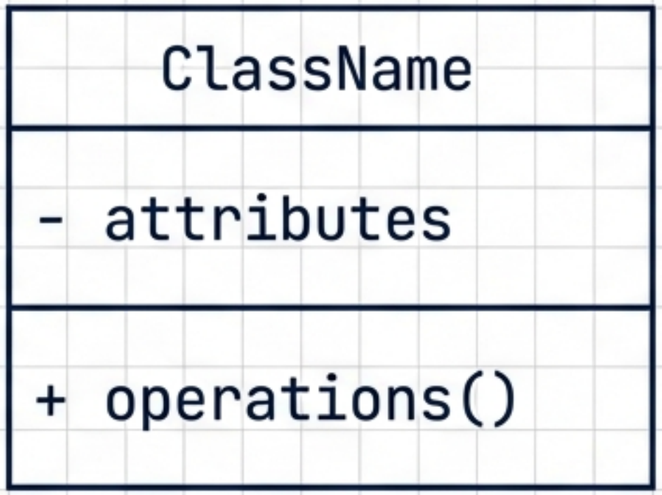


Realization (Interface)

IS-A relationship. Implements a contract.

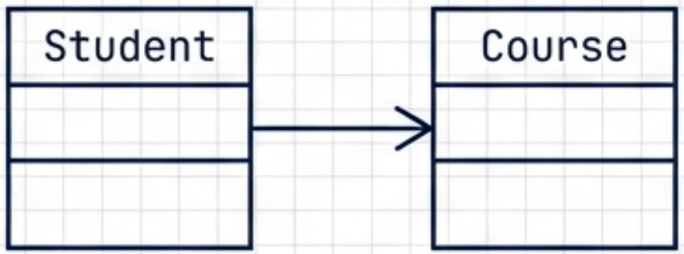
Dependency

USES relationship. Temporary, method-level parameter.



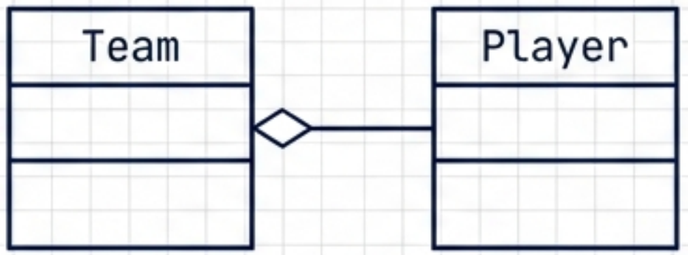
Defining Ownership: The 'Has-A' Hierarchy

Association



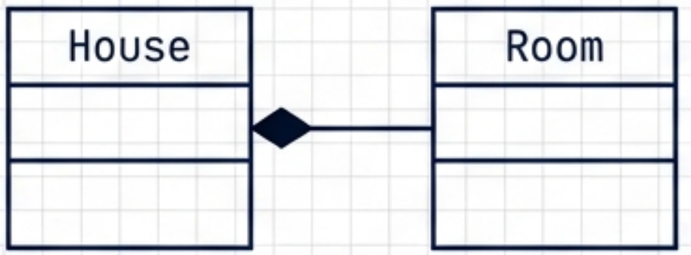
No Ownership.
Objects know each other, but exist independently.

Aggregation



Weak Ownership.
The part (Player) can outlive the whole (Team).

Composition



Strong Ownership.
The part (Room) dies if the whole (House) dies.

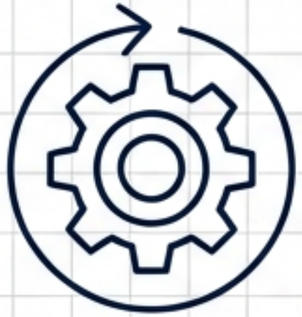
Prefer Association unless lifetime management is explicitly required.

The First Law: Single Responsibility Principle (SRP)

Inter Regular

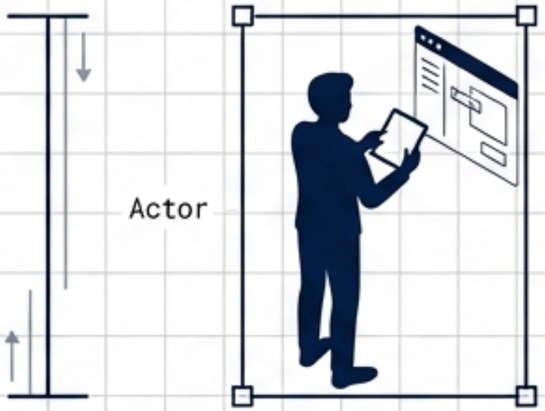
> **“A module should have one, and only one, reason to change.”**

JetBrains Mono



The Misconception

It is NOT about doing “one function”.



The Reality

It is about serving ONE ACTOR.

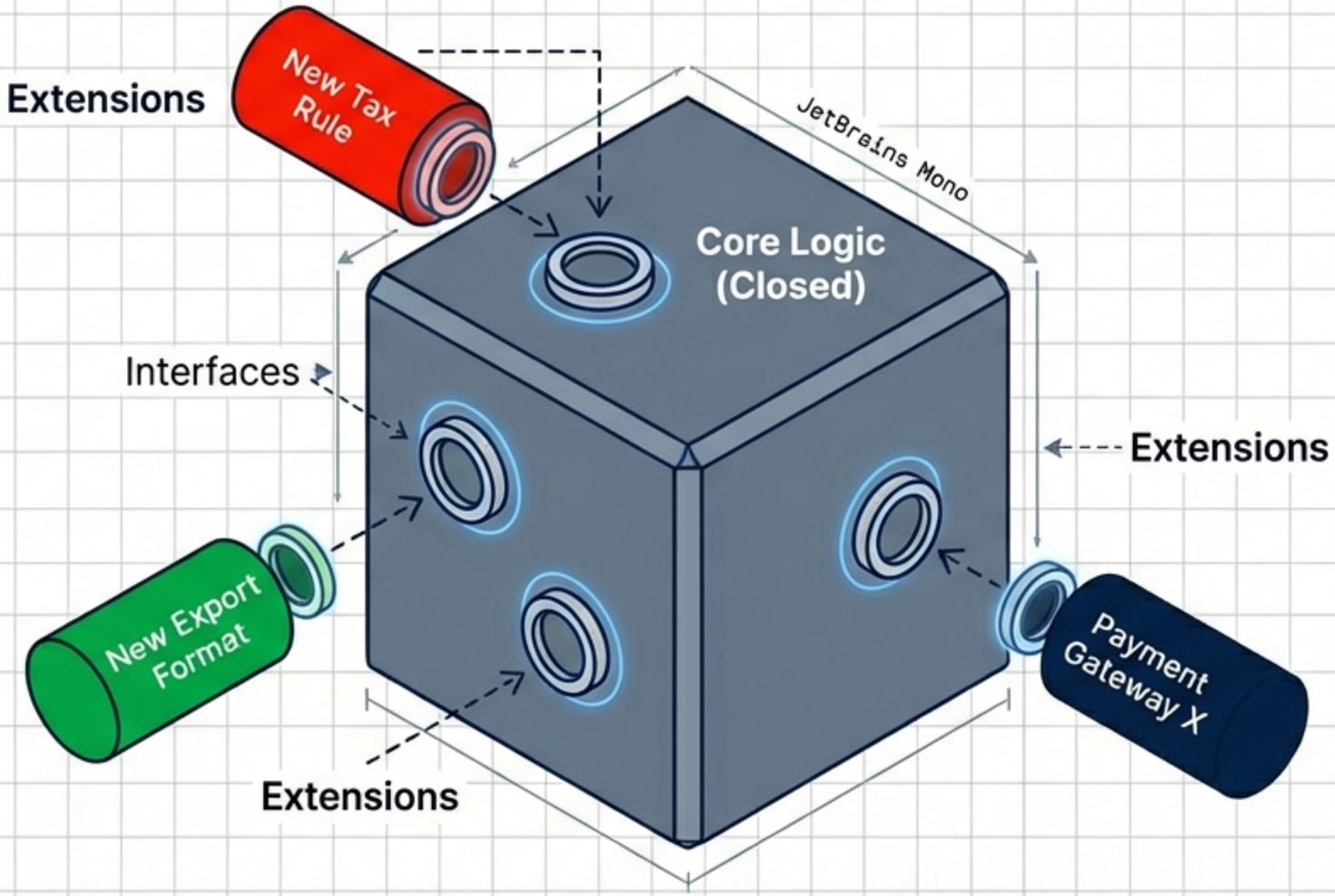
Metrics for SRP:

- **COHESION** (Target: High): Things that change together, stay together.
- **COUPLING** (Target: Low): Unrelated things are kept apart.

Rule: If SRP conflicts with **DRY** (Don’t Repeat Yourself), prioritize SRP.

Managing Change: Open-Closed Principle (OCP)

Open for Extension, Closed for Modification.



Inter Regular

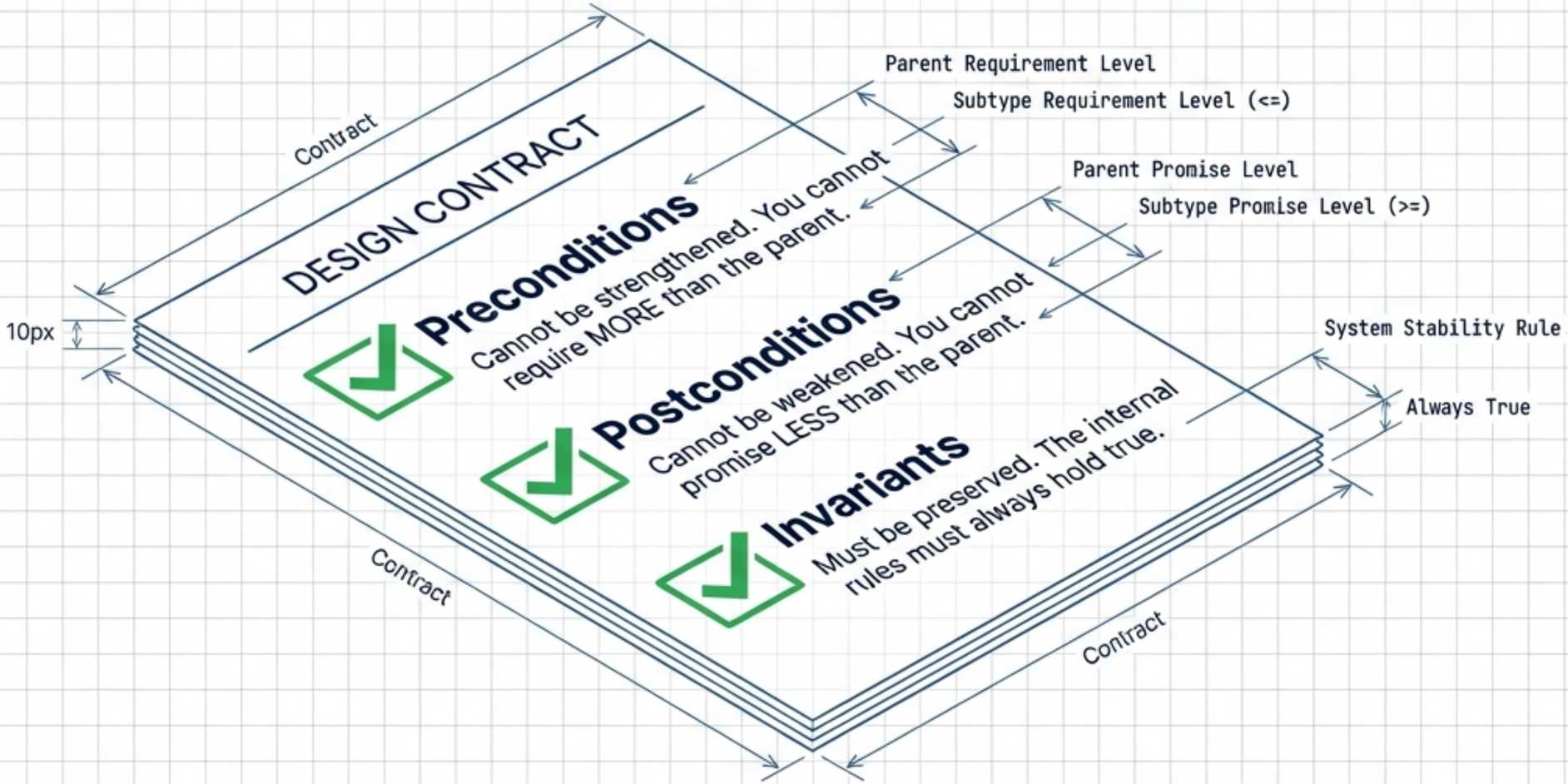
Mechanism: **Polymorphism** & Interfaces.

Goal: Add new code without touching old code.

⚠ Ousterhout Warning: Do not apply blindly. Wait for variation to occur to avoid ‘over-generalization’.

The Trust Protocol: Liskov Substitution Principle (LSP)

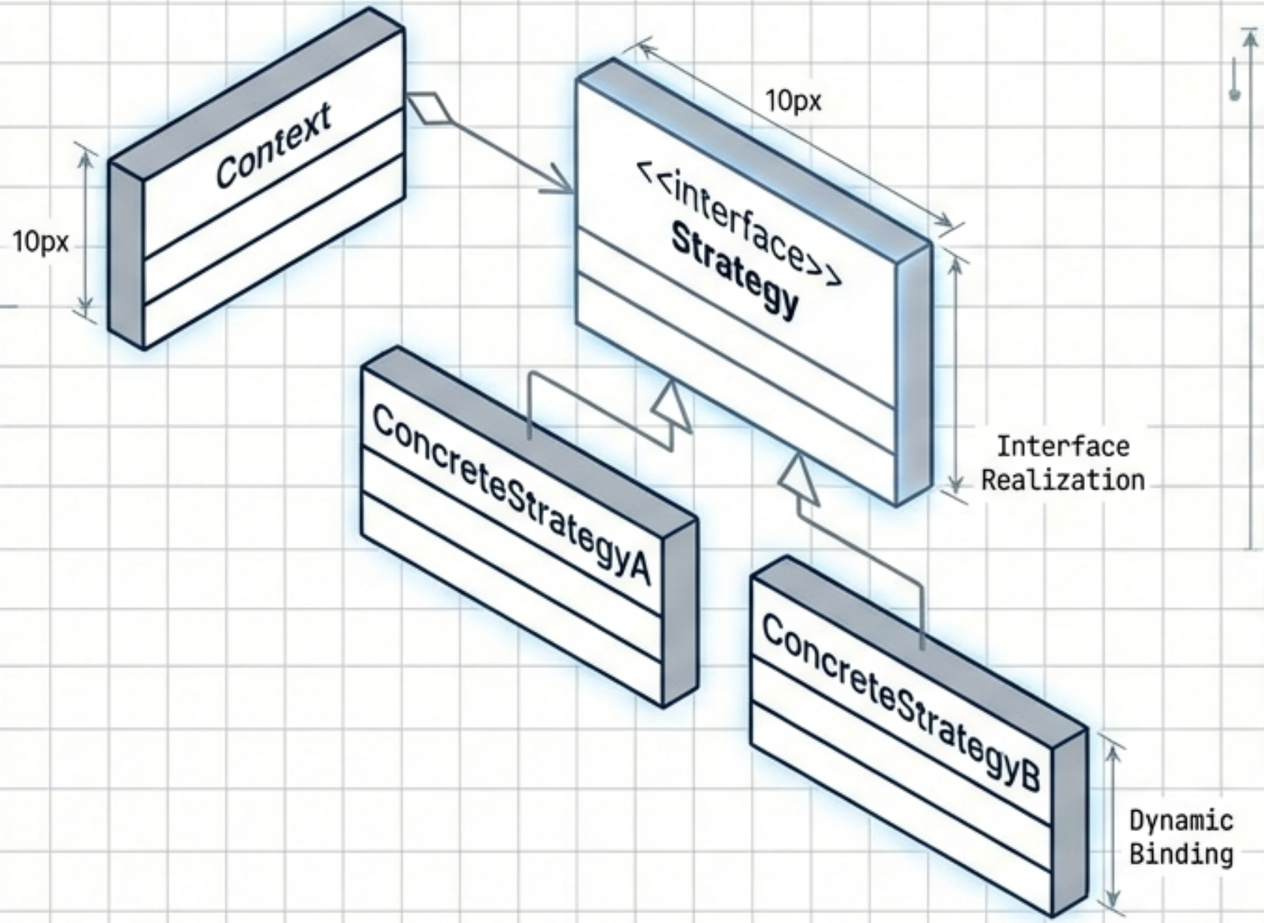
Subtypes must be substitutable for their base types without surprising the client.



⚠ LSP is about BEHAVIOR, not just compilation.

Weapon 1: The Strategy Pattern

Encapsulate interchangeable algorithms behind a stable interface.



Inter Regular

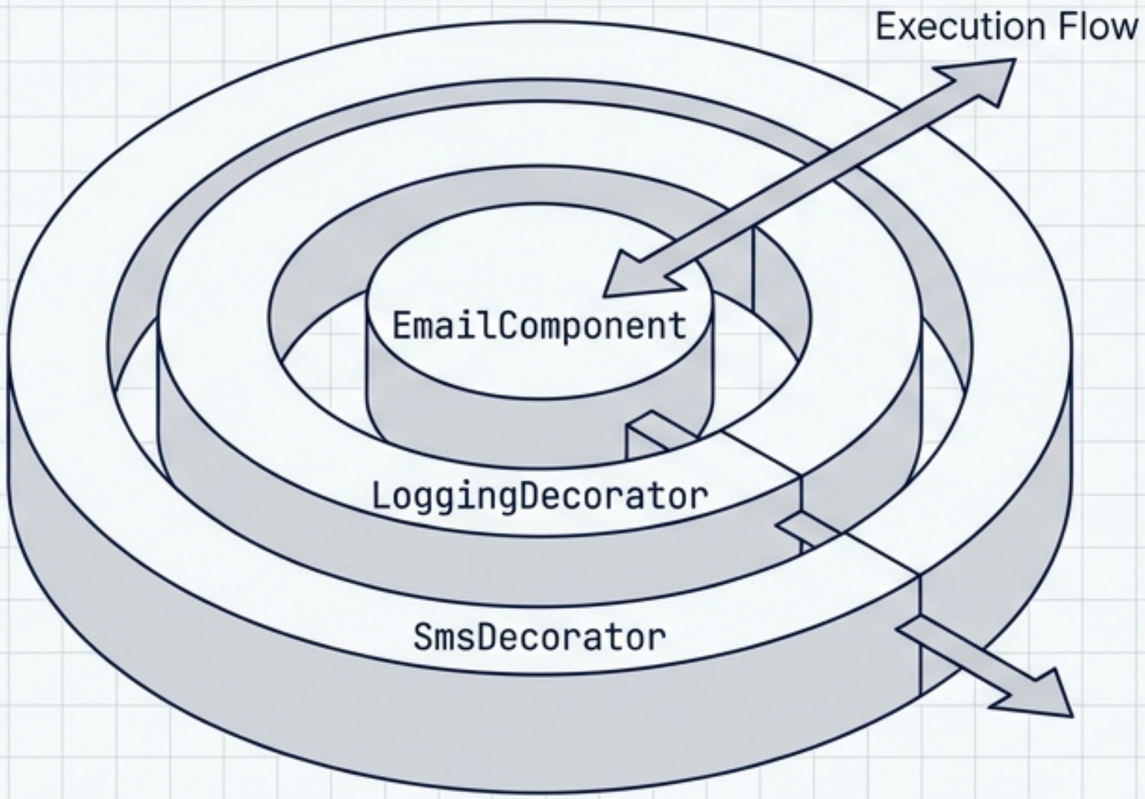
```
Context context = new Context();
context.setStrategy(new CreditCardStrategy());
context.pay(100.00);

// Later...
context.setStrategy(new PayPalStrategy());
context.pay(100.00);
```

⚠ The Context doesn't know HOW payment happens, only THAT it happens.

Weapon 2: The Decorator Pattern

Intent: Add responsibilities dynamically by wrapping objects.



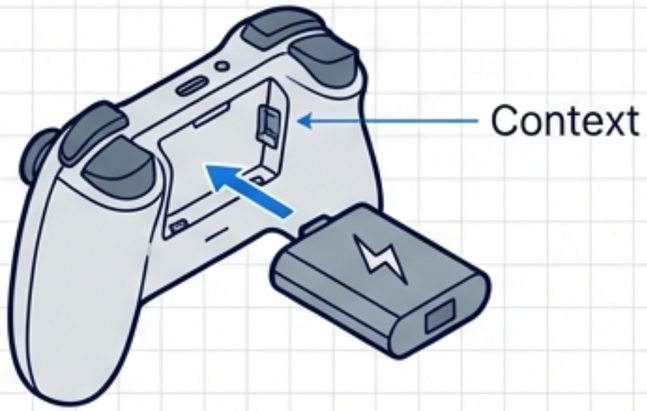

```
INotifier service = new Sms(
    new Logging(
        new Email()
    )
);
service.send(msg);
```

⚠ Inheritance used for Type Matching.
Composition used for Behavior Addition.

Swiss Engineering Blueprint

Distinguishing Your Weapons

Strategy Pattern



Goal: Change the guts.

Cardinality: Choose ONE at a time.

Relationship: Context HAS-A Strategy.

Decorator Pattern

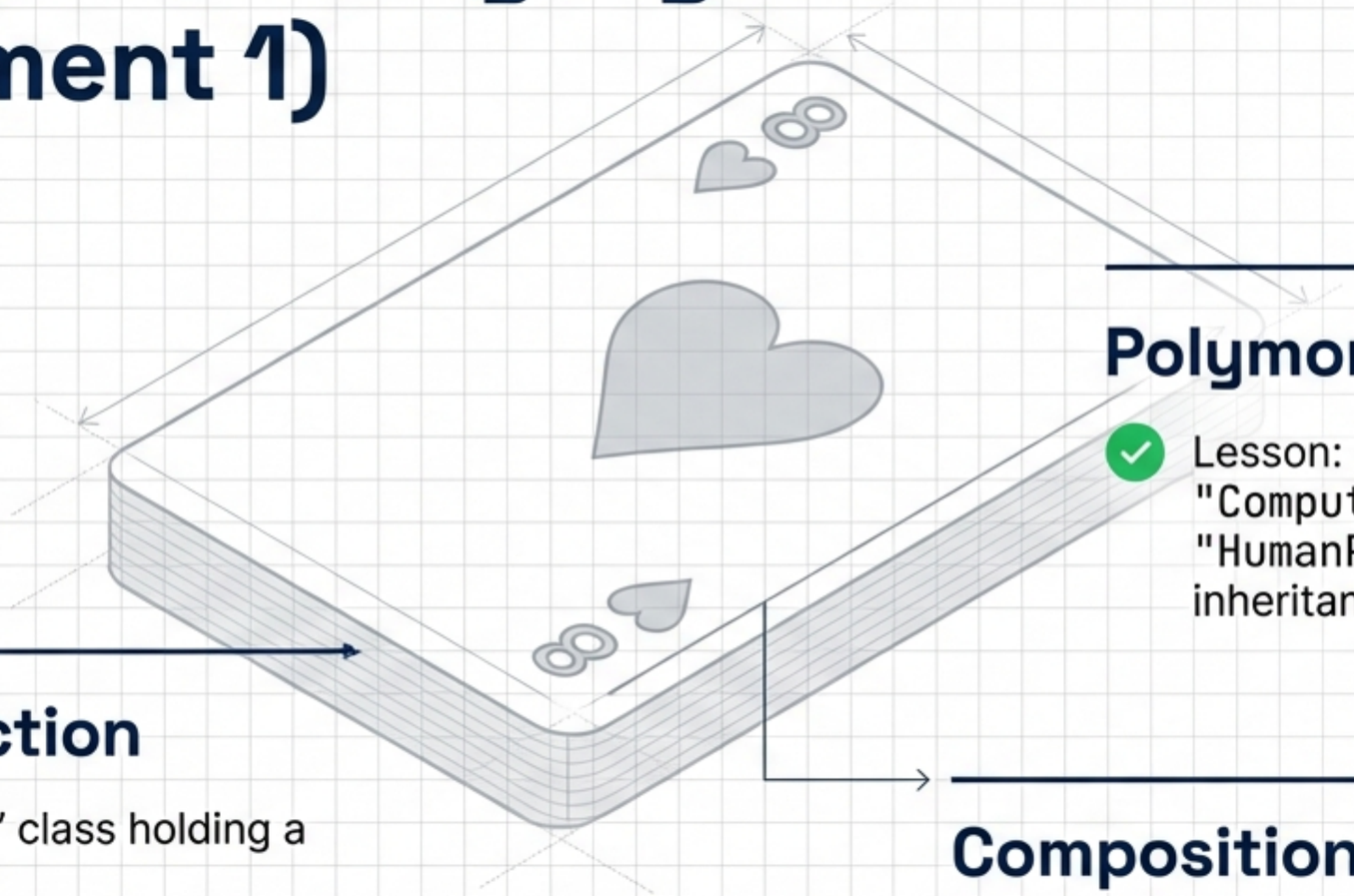
Goal: Change the skin / wrappers.

Cardinality: Stack MANY at once.

Relationship: Decorator IS-A Component AND HAS-A Component.

Exam Tip: If it's "A OR B", use Strategy. If it's "A AND B", use Decorator.

Field Report: Crazy Eights (Assignment 1)



Polymorphism

Lesson: "Player" class treats "ComputerPlayer" and "HumanPlayer" identically via inheritance.

State Protection

Problem: “Hand” class holding a list of cards.

Solution: Never expose the raw "List<Card>". Return a read-only copy.

Composition Root

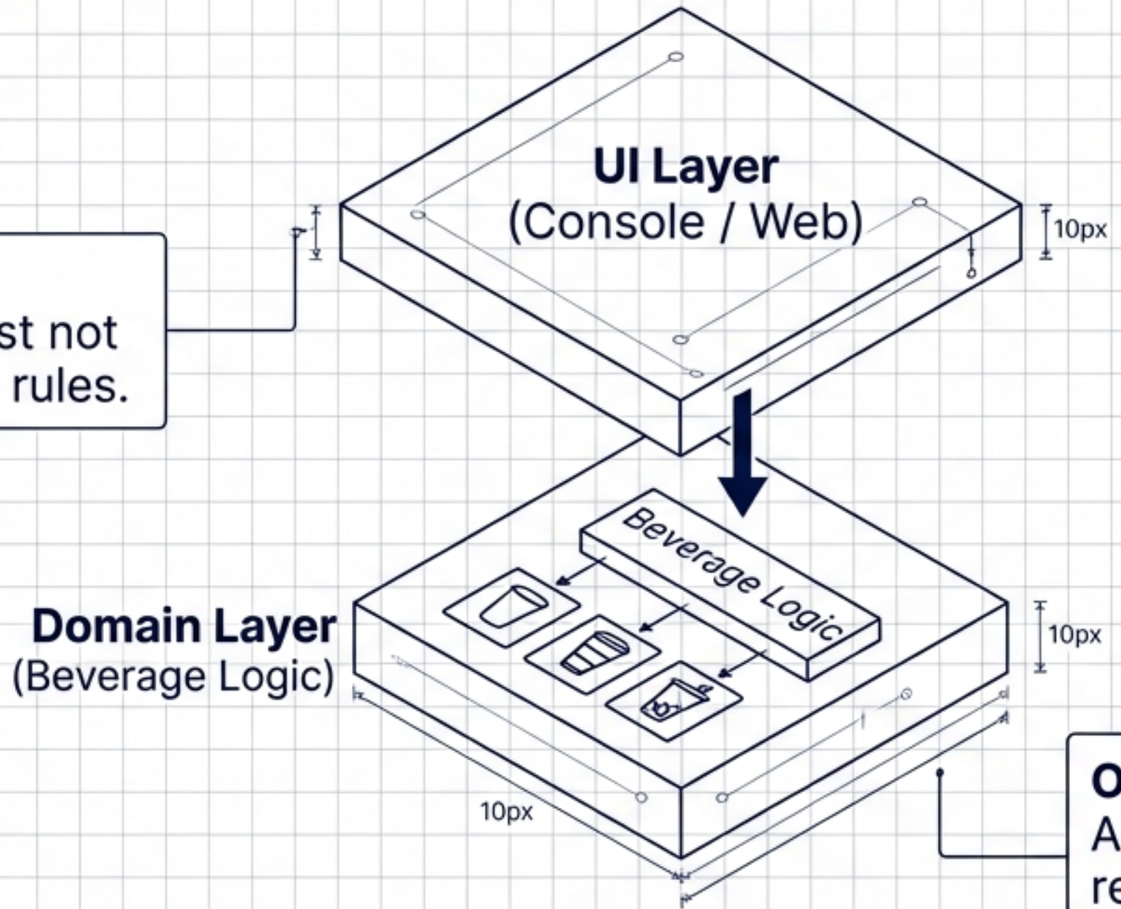
Lesson: Keep "Main" simple. Wire dependencies once, then run.

10px

Swiss Engineering Blueprint

Field Report: Tea Shop (Assignment 2)

Layering:
UI changes must not break business rules.



Pattern Application:
- **STRATEGY** used for Payment (Credit vs Cash).
- **DECORATOR** used for Customizations (Milk, Sugar, Boba).

OCP Win:
Adding a new topping does not require opening the 'Beverage' class.

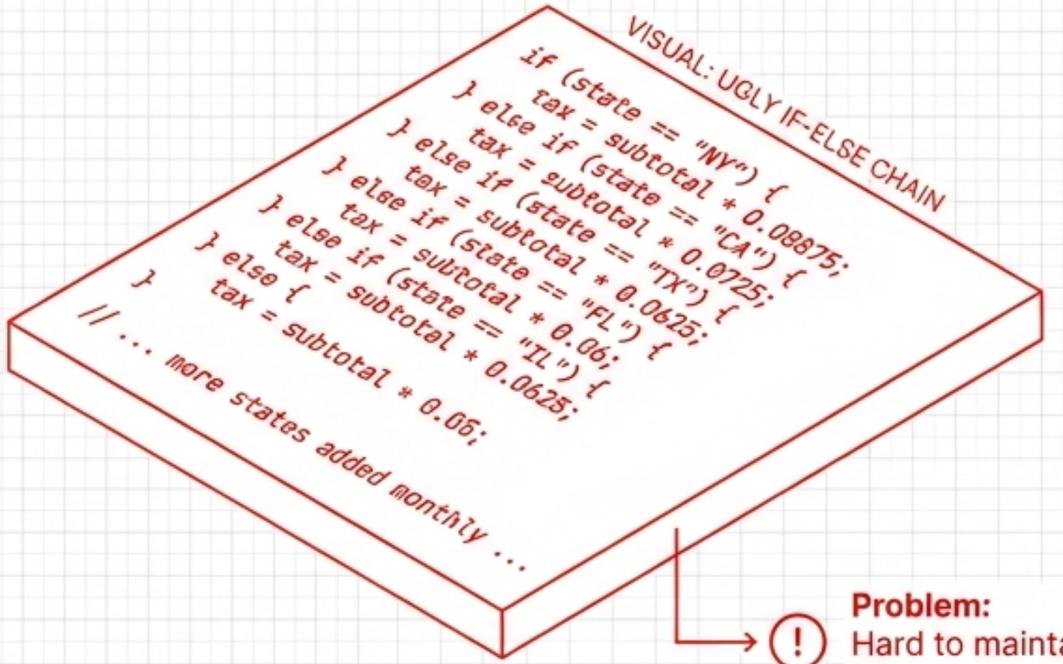
10px

Scenario Analysis: The Tax Problem

The Problem

Scenario: CheckoutService has a growing if/else block for tax rules by state.

Constraint: New rules arrive monthly.

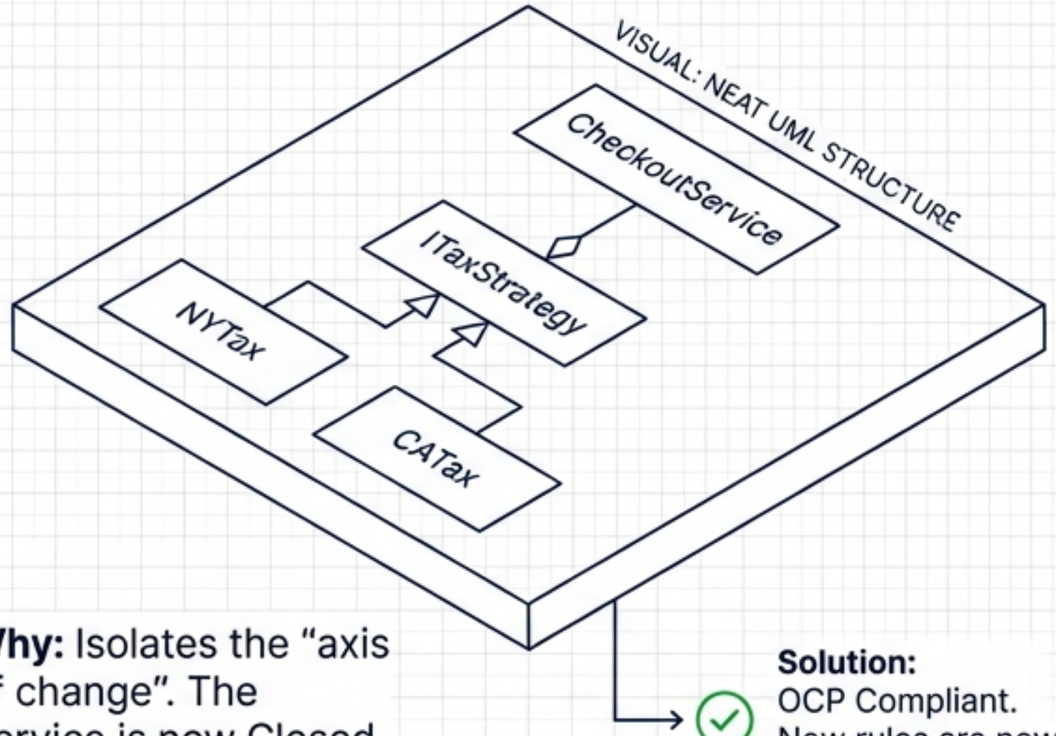


Problem: Hard to maintain, violates OCP.

The Solution

Question: Which refactor best supports OCP?

Answer: Introduce TaxStrategy.



Why: Isolates the "axis of change". The Service is now Closed for Modification.

Solution: OCP Compliant. New rules are new classes.

Scenario Analysis: The Layering Problem

The Scenario: You need to keep the same notification behavior but optionally add logging, retry, and metrics in different combinations at runtime.

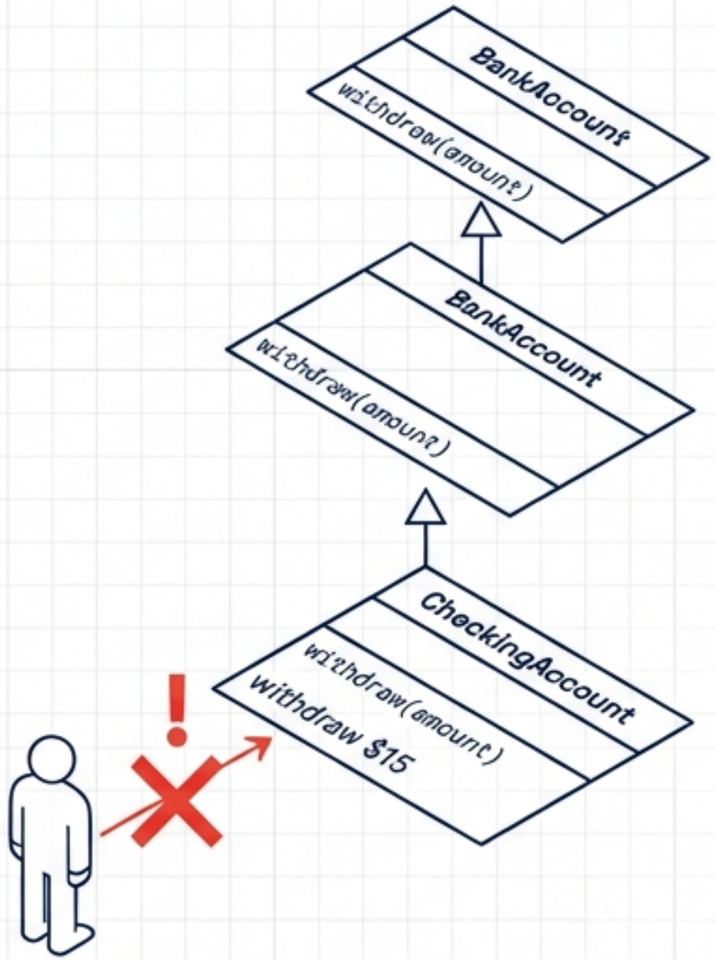
The Question: Strategy or Decorator?

DECORATOR

- **Strategy** is for choosing **ONE** algorithm.
- **Decorator** is for **combining MULTIPLE** orthogonal behaviors.
- **Key Clue**: “**Combinations**” and “**Add**”.

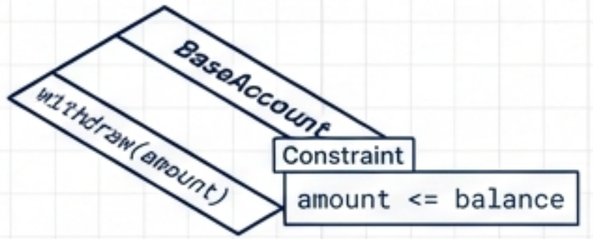
10px

Scenario Analysis: The Broken Promise



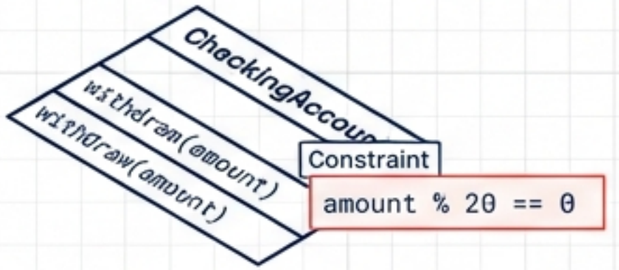
The Contract

Base Class: allows withdraw(amount) for any amount <= balance.



The Violation

Subtype: Adds rule "amount must be a multiple of 20".



The Verdict

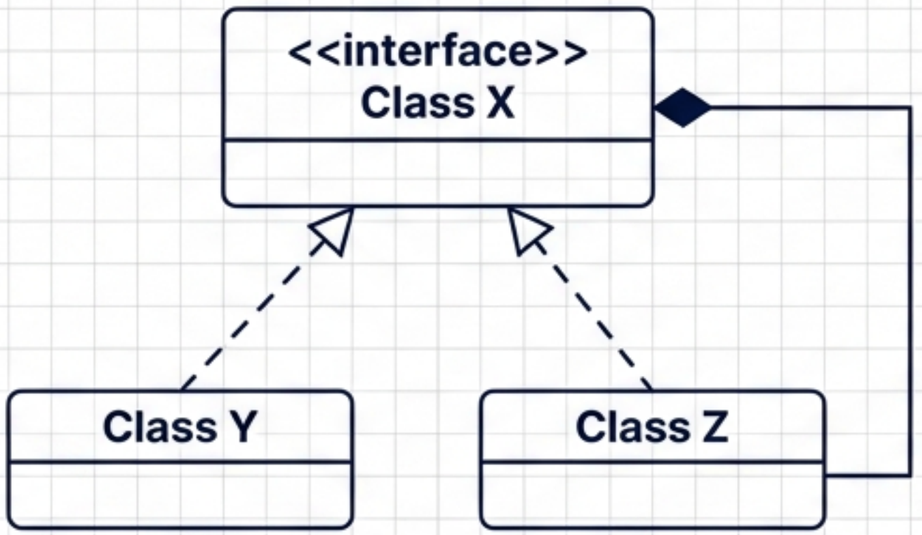
LSP VIOLATION

Reason: Strengthened Precondition.
Explanation: The client expects to be able to withdraw $15. The subtype rejects it. This is a surprise to the client.

10px

Swiss Engineering Blueprint

Visual Analysis: Identifying the Pattern



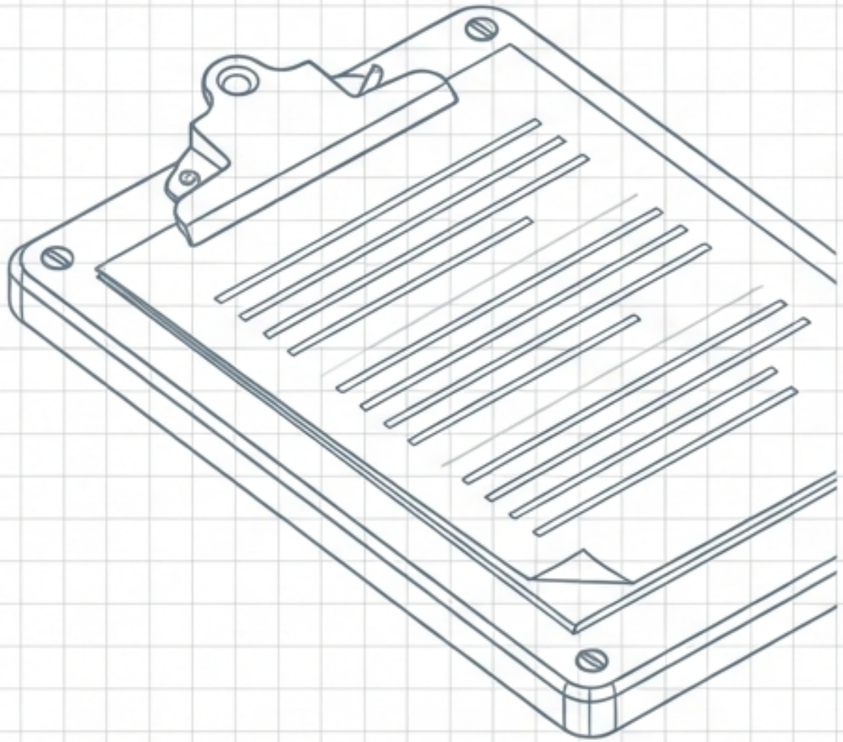
Question: Strategy or Decorator?

Answer: DECORATOR.

The Clue: The Recursive Relationship.

A Decorator **IS-A** Component (Implementation) AND **HAS-A** Component (Aggregation/Association).

Summary of Operations



- [x] **UML:** Can you distinguish Composition (filled diamond) vs. Aggregation (empty diamond)?
- [x] **Principles:** Can you spot an **SRP** violation? Can you explain **WHY** a subclass breaks **LSP**?
- [x] Patterns: Do you know when to use **Strategy** (swap) vs. **Decorator** (layer)?

"The best designs allow most developers to work without touching deep complexity." - Ousterhout

Final Deployment Orders

Don’t just memorize the definitions.
Look for the **RELATIONSHIPS.**
Look for the **COST OF CHANGE.**

Good luck on Exam 1. Design thoughtfully.