1. OOPS: Liskov Substitution Principle (LSP)

The Liskov Substitution Principle is the 'L' in SOLID. It states that **objects of a superclass should be replaceable with objects of its subclasses without breaking the application.**

Key Concept: Behavioral Subtyping

It's not just about matching method signatures; it's about matching **behavioral expectations**. If a parent class can do something, the child class must be able to do it in a way that doesn't surprise the user of the parent class.

- **Pre-conditions:** Cannot be strengthened in a subclass (you can't require more than the parent).
- **Post-conditions:** Cannot be weakened in a subclass (you can't promise less than the parent).

2. OOPS Scenario: The Rectangle-Square Problem

Problem: You have a `Rectangle` class with `set_width()` and `set_height()`. You create a `Square` class that inherits from `Rectangle`.

The Violation: In a `Square`, setting the width automatically changes the height. If a function expects a `Rectangle` and tries to double the width while keeping the height the same, it will fail if passed a `Square` object. The `Square` breaks the "contract" of the `Rectangle`.

The LSP Solution:

1. **Refactor the Hierarchy:** Don't force a "Square is-a Rectangle" relationship if the behaviors conflict.
2. **Abstract Base:** Create a `Shape` interface with a `get_area()` method.
3. **Independent Implementation:** Both `Rectangle` and `Square` implement `Shape` but handle their own dimensions internally.

3. Programming: Majority Element Problem

Problem: Find the element that appears more than $n/2$ times in an array of size n.

Approach 1: Hashing

- Count frequencies using a Hash Map.
- **Complexity:** $O(n)$ Time and $O(n)$ Space.

Approach 2: Boyer-Moore Voting Algorithm (Optimized)

This is the most efficient way to solve this, using $O(1)$ extra space.

Logic:

1. Initialize a `candidate = None` and a `count = 0`.
2. Traverse the array:
   - If `count == 0`: Set the current element as the `candidate`.
   - If current element == `candidate`: `count += 1`.
   - Else: `count -= 1`.
3. The remaining `candidate` is the majority element.

Why it works: Since the majority element appears more than half the time, it will always "outvote" all other elements combined, eventually surviving the count increments and decrements.

- **Time Complexity:** $O(n)$
- **Space Complexity:** $O(1)$

4. SQL: Views

A **View** is a virtual table based on the result-set of an SQL statement. It contains rows and columns, just like a real table, but it doesn't store data itself.

Why use Views?

1. **Simplicity:** You can wrap a query with 5 complex `JOINs` into a single view called `CustomerOrderSummary`. Users can then query the view as if it were a simple table.
2. **Security:** You can create a view that shows employee names and departments but **hides** their sensitive `salary` or `ssn` columns.
3. **Consistency:** If the underlying table structure changes, you only update the view definition. The reports and apps using that view don't need to change their queries.

Syntax Example:

```
CREATE VIEW active_sales_summary AS
SELECT p.product_name, SUM(s.quantity) AS total_sold
FROM products p
JOIN sales s ON p.id = s.product_id
WHERE s.sale_date > '2023-01-01'
GROUP BY p.product_name;

-- Now you can simply call:
SELECT * FROM active_sales_summary WHERE total_sold > 100;
```

Summary Table

Topic	Focus	Key Takeaway
OOPS	Liskov Substitution	Subclasses must honor the behavioral contract of the parent.

DSA	Majority Element	Boyer-Moore algorithm achieves $O(n)$ time with $O(1)$ space.
SQL	Views	Virtual tables provide abstraction, security, and simpler query interfaces.

Day 20 complete! You've reached the one-third mark of the journey. Your understanding of robust system design is growing every day.