1. OOPS: Interface Segregation Principle (ISP)

The Interface Segregation Principle is the 'I' in SOLID. It states that **no client should be forced to depend on methods it does not use.**

The Core Idea: "Thin" Interfaces

Instead of creating one "Fat" interface that contains every possible method, we should split it into smaller, more specific "Thin" interfaces. This ensures that classes only implement the functionality that is relevant to them.

- **Signs of ISP Violation:** A class implements an interface but leaves several methods empty or throws a `NotImplementedException` .
- **The Benefit:** Reduces the impact of changes. If a method in a "Scanner" interface changes, it won't affect classes that only care about "Printing."

2. OOPS Scenario: The Multi-Function Printer

Problem: You have an interface `IMultiFunctionDevice` with methods `print()` , `scan()` , and `fax()` . You want to create a `SimplePrinter` class.

The Violation: A `SimplePrinter` cannot scan or fax. To satisfy the compiler, you have to write:

```
class SimplePrinter(IMultiFunctionDevice):
    def print(self):
        # actual logic
    def scan(self):
        raise NotImplementedError("I can't scan!")
    def fax(self):
        pass # empty method
```

This is a "Polluted Interface." Any code that uses `IMultiFunctionDevice` might try to call `scan()` on your printer and crash.

The ISP Solution: Split the interface into `IPrinter` , `IScanner` , and `IFax` .

1. `SimplePrinter` implements only `IPrinter` .
2. `AdvancedOfficeMachine` implements all three.
3. **The Result:** The system is cleaner, more type-safe, and easier to test.

3. Programming: First Missing Positive

This is a "Hard" category problem because it requires $O(n)$ time and $O(1)$ auxiliary space.

Problem: Given an unsorted integer array, find the smallest missing positive integer.

The Algorithm: Cyclic Sort (In-Place)

The key insight is that the missing positive must be in the range $[1, n+1]$. We try to place every number x at index $x-1$.

1. **Rearrange:** Iterate through the array. While `1 <= arr[i] <= n` and `arr[i] != arr[arr[i]-1]` :
   - Swap `arr[i]` with `arr[arr[i]-1]` .
2. **Identify:** Iterate through the rearranged array. The first index `i` where `arr[i] != i + 1` tells us that `i + 1` is the missing number.
3. **Fallback:** If all numbers are in place, the missing number is `n + 1` .

Why it's efficient:

- **Time:** $O(n)$. Although there is a `while` loop inside a `for` loop, each element is moved to its correct position at most once.
- **Space:** $O(1)$ because we modify the input array directly.

4. SQL: Stored Procedures

A **Stored Procedure** is a prepared SQL code that you can save and reuse. Think of it as a "Function" for your database.

Why use Stored Procedures?

1. **Performance:** They are compiled once and stored in the database, reducing the overhead of parsing and optimizing the query every time it's run.
2. **Reduced Network Traffic:** Instead of sending a long query over the network, you just send the procedure name and parameters.
3. **Security:** You can give users permission to execute a procedure without giving them direct access to the underlying tables. This prevents SQL injection and accidental data deletion.

Syntax Example:

```sql
CREATE PROCEDURE GetCustomerOrders
    @CustomerID INT,
    @MinAmount DECIMAL(10,2)
AS
BEGIN
    SELECT o.order_id, o.order_date, o.total_amount
    FROM orders o
    WHERE o.customer_id = @CustomerID AND o.total_amount > @MinAmount;
END;

-- To execute:
EXEC GetCustomerOrders @CustomerID = 5, @MinAmount = 500.00;
```

Summary Table

Topic	Focus	Key Takeaway
OOPS	Interface Segregation	Prefer small, specific interfaces over large, generic ones.
DSA	First Missing Positive	Use the array indices as a "hash" to sort numbers in $O(n)$ time and $O(1)$ space.
SQL	Stored Procedures	Encapsulates logic at the DB level for better performance and security.

Day 21 complete! You're mastering the nuances of professional architecture and high-performance algorithms.