1. OOPS: Factory Pattern vs. Singleton Pattern

Both are **Creational Design Patterns**, but they solve completely different problems regarding how objects are born in your system.

Comparison Table

Feature	Singleton Pattern	Factory Pattern
Intent	Ensures **only one** instance of a class exists.	Provides an interface to create objects without specifying the exact class.
Focus	Controlling **access** to a shared resource.	Controlling the **process** of object creation.
Instance Count	Always exactly one.	Can create many instances of different types.
Global Access	Provides a global entry point to the instance.	No global instance; returns a new instance as requested.
Testing	Hard to test (Global state persists).	Easy to test (Mock objects can be returned).

2. OOPS Scenario: The Singleton Dependency Trap

Problem: A developer uses a Singleton `ConfigurationManager` everywhere in the code. Later, they find that writing Unit Tests is impossible because the "Global State" of the configuration from one test leaks into the next one.

The Hidden Dependency: When Class A calls `Singleton.getInstance()`, Class A is now "hiddenly" dependent on that global object. You cannot test Class A in isolation without also setting up the entire Singleton state.

The Factory Solution: By refactoring to a **Factory**, the code doesn't "reach out" for a global instance. Instead:

1. The Factory is passed into Class A (Dependency Injection).
2. Class A asks the Factory: "Give me the configuration I need."
3. During testing, we can pass a **Mock Factory** that returns a fake configuration, making the code flexible, testable, and loosely coupled.

3. Programming: Hashing + Sliding Window

This is a more advanced version of the sliding window technique from Day 5, using a Hash Map to track character positions.

Problem: Longest Substring Without Repeating Characters.

The Optimized Algorithm:

1. Initialize two pointers, `left = 0` and `right = 0`.
2. Use a Hash Map to store the **last seen index** of each character.
3. Traverse the string with `right`:
   - If the current character is already in the map **and** its index is $\geq left$:
     - Move `left` to `map[char] + 1` (Jump past the previous occurrence).
   - Update the character's position in the map.
   - Calculate the window length (`right - left + 1`) and update the `max_length`.
4. **Complexity:** $O(n)$ Time (Single pass) and $O(\min(m, n))$ Space.

   Why the jump? Instead of moving `left` one by one, the Hash Map allows us to "teleport" the window past the repeated character instantly.

4. SQL: Subqueries & Constraints

Constraints are rules enforced on data columns to ensure the **Integrity** and **Accuracy** of the database.

Core Constraints:

- **PRIMARY KEY:** Uniquely identifies each record. Cannot be NULL. (One per table).
- **FOREIGN KEY:** Prevents actions that would destroy links between tables (Referential Integrity).
- **UNIQUE:** Ensures all values in a column are different. (Can have multiple per table, and allows one NULL).
- **NOT NULL:** Ensures a column cannot have a NULL value.
- **CHECK:** Ensures that the value in a column meets a specific condition (e.g., `Age > 18`).

Subqueries in Filtering:

Subqueries are often used in `WHERE` or `HAVING` clauses to filter data based on dynamic values.

Syntax Example: Find products whose price is greater than the average price of all products.

```
SELECT product_name, price
FROM products
WHERE price > (SELECT AVG(price) FROM products);
```

Summary Table

Topic	Focus	Key Takeaway
OOPS	Singleton vs. Factory	Singleton limits instances; Factory abstracts the "how" of creation.
DSA	Hashing + Sliding Window	Map-based jumping reduces window adjustments to $O(1)$, keeping the total time $O(n)$.
SQL	Constraints	Essential for preventing "Garbage In, Garbage Out"; subqueries enable dynamic logic.

Day 18 complete! You've moved from basic patterns to identifying architectural traps