1. OOPS: Polymorphism (Runtime Behavior)

While Day 8 introduced the concept, Day 22 focuses on the underlying mechanics of **Runtime Polymorphism** (also known as Dynamic Method Dispatch).

How it Works:

1. **Method Overriding:** A subclass provides a specific implementation of a method that is already defined in its parent class.
2. **Upcasting:** A parent class reference points to a child class object (e.g., `Animal myDog = new Dog();`).
3. **Late Binding:** The compiler does not know which method to call at compile-time. Instead, the Java Virtual Machine (JVM) or Python Interpreter looks at the **actual object type** at runtime to decide which method to execute.

The Power of Runtime Behavior:

It allows you to write code that treats different objects in a uniform way while still allowing each object to behave according to its specific type. This is the foundation of "Plug-and-Play" architecture.

2. Programming: Nearest Greater Element

This problem introduces the **Monotonic Stack**, a powerful pattern used to solve many $O(n^2)$ array problems in $O(n)$ time.

Problem: For each element in an array, find the first element to its right that is strictly greater. If none exists, return -1.

Approach 1: Brute Force

- For every element, use an inner loop to scan the right side.
- **Complexity:** $O(n^2)$.

Approach 2: Monotonic Stack (Optimized)

We traverse the array (usually from right to left) and maintain a stack of elements that could potentially be the "Greater Element" for indices to the left.

Algorithm (Right to Left):

1. Initialize an empty stack and a result array of size n.
2. Traverse the array from $n-1$ down to 0:
    - While the stack is not empty AND `stack.top() <= arr[i]` :
        - Pop from the stack (these elements are smaller and "blocked" by `arr[i]`).
    - If the stack is empty, `result[i] = -1` .

- Else, `result[i] = stack.top()`.
- Push `arr[i]` onto the stack.

3. Return the result array.

Complexity:

- **Time:** $O(n)$. Even with the nested `while` loop, each element is pushed and popped exactly once.
- **Space:** $O(n)$ for the stack.

3. SQL: SELF JOIN

A **SELF JOIN** is a regular join, but the table is joined with itself. It is essential for querying hierarchical data stored in a single table.

Scenario: Employee-Manager Hierarchy

In many databases, an `Employees` table has a `manager_id` column that refers back to the `employee_id` in the same table.

Syntax Example:

To list all employees along with the name of their manager:

```
SELECT
    e.employee_name AS Employee,
    m.employee_name AS Manager
FROM employees e
INNER JOIN employees m ON e.manager_id = m.employee_id;
```

Key Logic:

- **Aliases are Mandatory:** You must give the table two different aliases (e.g., `e` for employee and `m` for manager) so the database can distinguish between the two instances of the same table.
- **Use Cases:**
  - Organizational hierarchies (Manager/Subordinate).
  - Pre-requisite structures (Course/Pre-req).
  - Sequential data (finding rows where a value increased compared to the previous day).

Summary Table

Topic	Focus	Key Takeaway
OOPS	Runtime Polymorphism	Behavior is determined by the object type at runtime, enabling dynamic dispatch.

DSA	Nearest Greater Element	Monotonic Stacks reduce search complexity from $O(n^2)$ to $O(n)$.
SQL	SELF JOIN	Uses table aliases to relate records within a single hierarchical table.