---

1. Greenhouse/Garden Area: The Engine of Life 🌿

This area is essentially your core data and simulation engine. It's where the raw materials (plants) are managed, grow, and are prepared for sale.

Key Concept	Practical C++ Implementation Goal	Design Pattern Hint (Why?)
Plant Care	Model diverse behaviors (watering, sunlight) without changing the main `Plant` class code.	Strategy Pattern (To define interchangeable families of plant care algorithms: `WateringStrategy`, `SunlightStrategy`).
Plant Life Cycle	Track state changes (Seedling Juvenile	State Pattern (To let a plant object alter its behavior when its internal state changes, like transitioning from "Growing" to "ForSale").

	Mature/Ready-to-Sell) and trigger events (e.g., alert staff when mature).	
Inventory & Stock	A centralized system to hold all plant objects and allow staff to query, add, or remove stock efficiently.	Singleton Pattern (To ensure there's only one official `Inventory` or `GreenhouseManager` object accessible globally).

---

2. The Staff: The Connecting Force

The staff act as agents that perform actions and facilitate communication between the plants (Greenhouse) and the customers (Sales Floor). They bring the system to life by performing tasks.

Key Concept	Practical C++ Implementation Goal	Design Pattern Hint (Why?)
Plant Care	Allow a `Staff` object to execute various care routines (water, fertilize, prune) on different `Plant` objects.	Command Pattern (To encapsulate requests as objects, such as `WaterPlantCommand`, `PrunePlantCommand`, which can be queued, logged, or easily executed by a staff member).
Staff Coordination	Ensure staff are alerted when a plant is mature, a customer asks a	Observer Pattern (The `Staff` (Observer) watches the `Plant` or `Inventory` (Subject). When a plant matures, the subject

	question, or inventory is low.	notifies all staff observers).
Role Extension	Easily add new staff roles (e.g., a `Landscaper` or `GreenhouseManager`) with different, specialized abilities.	Decorator Pattern or Factory Method (Decorator can add responsibilities to a base `Staff` object dynamically; Factory Method can instantiate the correct staff role).

---

3. The Customers and The Sales Floor: The Revenue Stream

The sales floor models user interaction and how dynamic requests (browsing, customization) are fulfilled by the available stock and staff advice.

Key Concept	Practical C++ Implementation Goal	Design Pattern Hint (Why?)
Personalisation	Allow a customer to add features (like a fancy pot or gift wrap) to a standard `Plant` object before purchase.	Decorator Pattern (To wrap a base `Plant` object with optional features like `DecorativePotDecorator` or `GiftWrapDecorator`, modifying the final price and description).

Interaction & Sales	**A central agent (`SalesManager` or `Staff`) handles customer requests, validates inventory, and completes the sale.**	**Facade Pattern (A `SalesFacade` provides a simple interface—e.g., `processPurchase(Customer, Plant, Customizations)`—that hides the complexity of checking inventory, applying customizations, and updating stock).**
Customer Browsing	**Structure the plant inventory so customers can easily navigate or staff can efficiently search (e.g., "all succulents," "plants needing low light").**	**Composite Pattern (To treat individual plants and groups of plants/shelves uniformly, allowing customers to browse a "shelf" or a "whole section" recursively).**

Important Dates:

- 6 October - submit prac 5 (planning and stuff) with tutor
- 20 October - make necessary changes and implementation
- 27 October - Working implementation and begin demo prep
- 2 November - submit report and diagrams on Clickup and code on Fitchfork
- 3-5 November - demo

Important Details:

- At least 10 patterns
- Doxygen needs to be used

Three parts to the Prac

1. **Nursery or Greenhouse**
2. **Staff**

3. Customers

Task 1 (6 diagrams and functional requirements)

Functional and Non Functional Requirements

For every pattern we need a functional requirement
At least 3 non functional requirement

Greenhouse / Plant System

1. **FR1 – Plant Propagation:** The system must allow propagation of plants through *seeding, cloning, or grafting* methods. *(Factory Method / Prototype Pattern)*

2. **FR2 – Plant Growth Management:** The system must simulate the growth stages of plants (seedling → growing → mature). *(State Pattern)*

3. **FR3 – Care Strategy Application:** The system must apply different care strategies (watering, sunlight, fertilization) depending on the plant type. *(Strategy Pattern)*

4. **FR4 – Plant Generation:** The system must generate plants dynamically with varying properties (species, color, size). *(Builder or Abstract Factory Pattern)*

5. **FR5 – Inventory Tracking:** The system must track all plants currently growing, sold, or returned. *(Singleton Inventory + Observer Pattern)*

Customer and Sales

6. **FR6 – Customer Interaction:** The system must allow customers to browse available plants and request customizations. *(Mediator Pattern for staff-customer communication)*

7. **FR7 – Plant Customisation:** The system must allow customers to request decorative wrapping, potting, or arrangement. *(Decorator Pattern)*

8. **FR8 – Purchase and Order Management:** The system must process purchases and update stock accordingly. *(Command Pattern)*

9. **FR9 – Delivery Coordination:** The system must create and manage plant delivery orders using appropriate staff roles. *(Factory or Chain of Responsibility Pattern)*

Staff and Management

10. **FR10 – Staff Coordination:** The system must assign staff to specific tasks (customer assistance, propagation, delivery). *(Mediator / Observer Pattern)*

11. **FR11 – Role Specialization:** The system must support different staff roles (Manager, Normal Staff, Delivery Staff, Care Staff). *(Inheritance + Factory Pattern)*

12. **FR12 – Reporting and Monitoring:**
    The system must generate nursery status reports (inventory, sales, growth stats). *(Facade Pattern)*

13. **FR13 – Task Scheduling:** The system must synchronize greenhouse and customer floor activities to prevent conflicts. *(Command + Observer Pattern)*

System Operations

14. **FR14 – Data Persistence:** The system must maintain plant and transaction records for future simulation runs. *(Singleton + Memento Pattern optional)*

15. **FR15 – User Notifications:** The system must notify customers or staff when stock or plant status changes. *(Observer Pattern)*

⚙️ *Non-Functional Requirements*

NFR1 – Scalability

The system must handle an expanding nursery — allowing the addition of new plant types, staff roles, and customer features without major code restructuring. *(Addresses: Extensibility, Maintainability)*

NFR2 – Reliability

The system must ensure that all updates to plant inventory and sales records remain consistent even if an operation fails midway. *(Addresses: Data Integrity, Fault Tolerance)*

NFR3 – Performance

The system must process customer orders, staff updates, and plant growth simulations with minimal delay (<1 second per action in text-based mode). *(Addresses: Efficiency, Responsiveness)*

NFR4 – Usability

The system must provide a clear, intuitive text-based interface so users can easily navigate between greenhouse, customer, and staff interactions. *(Addresses: User Experience, Learnability)*

NFR5 – Modularity

All system components (plants, staff, customers, greenhouse) must be implemented as loosely-coupled modules to support reuse and independent development. *(Addresses: Maintainability, Reusability)*

UML Diagrams:

Class Diagram

State Diagram
At least 4 states and at least one state able to move forward and backwards

Activity Diagram
At least 4 steps

Sequence Diagram
At least 3 objects interacting

Object Diagram
Provide a snapshot of system at the given moment in time. At least 5 objects

Communication Diagram

We need to make sure the formatting is the same and the logic behind the different diagrams otherwise we will be penalised