Plant Nursery Simulator

Design Patterns, Requirements, and System Overview

The Tormentos (COS214 Final Project)

November 2025

Plant Nursery Simulator — Final Report

Design Patterns, Requirements, and System Overview

The Tormentos (COS214 Final Project)

November 2025

Contents

1. Research Brief: Nursery Domain Findings 1

1.1 Plant Care and Growth Management 1

1.2 Staff Coordination and Task Flow 1

1.3 Customer Experience and Sales Workflow 1

1.4 System Scalability and Data Management 1

1.5 Assumptions and Definitions 2

1.6 References 2

2. System Overview 2

3. Functional & Non-Functional Requirements 2

3.1 Functional Requirements (FR) 2

3.2 Non-Functional Requirements (NFR) 3

4. Design Patterns Report 3

4.1 Iterator — Inventory 4

4.2 Iterator — Greenhouse 5

1.4.3 Composite — Greenhouse Structure 6

1.4.4 Facade — Sales 7

1.4.5 Builder — Orders 8

1.4.6 Observer — Plants, Staff, Stock Items 9

1.4.7 Adapter — Inventory File I/O 10

1.4.9 State — Plant Lifecycle 12

1.4.10 Command — Plant Care Tasks 13

1.4.11 Mediator — Floor Coordination 14

1.4.12 Chain of Responsibility — Care Routing 15

1.4.13 Strategy — Watering and Fertilising 16

1.5 5. GUI Requirements 17

1.5.1 5.1 Home Dashboard 17
1.5.2 5.2 Greenhouse Visualisation 17
1.5.3 5.3 Plant Lifecycle Monitor 17
1.5.4 5.4 Inventory Management 17
1.5.5 5.5 File Import & Export (Adapters) 17
1.5.6 5.6 Sales & Customer Operations (Facade) 17
1.5.7 5.7 Prototype Registry & Plant Templates 17
1.5.8 5.8 Staff & Task Management 17
1.5.9 5.9 Care Strategy Management 17
1.5.10 5.10 Order & Return History 17
1.5.11 5.11 System Administration 17
1.5.12 5.12 Error Handling & Notifications 18
1.6 6. Plant Lifecycle State Machine 21
1.7 7. Pattern-to-Requirement Summary 21
1.8 8. Spec Compliance & Submission Checklist 22
1.9 9. References 22

1. Research Brief: Nursery Domain Findings

The research phase examined how commercial nurseries operate across plant cultivation, staff coordination, and customer experience. Nurseries behave like living ecosystems where biological growth, human scheduling, and sales interactions constantly influence each other. Understanding those loops guided the simulator so that it mirrors the adaptability of a real operation while staying maintainable.

1.1 Plant Care and Growth Management

- Nurseries manage diverse plant families (annuals, perennials, succulents, shrubs, trees) with distinct water, soil, nutrient, and sunlight profiles (Department of Agriculture, 2023; Royal Horticultural Society, 2022).
- Climate and irrigation systems must adjust on schedule to avoid stress across batches.
- **Insight applied:** Each `PlantInstance` owns pluggable watering and fertilising strategies (**Strategy**) so care algorithms can reflect cultivar differences. Lifecycle state transitions (Seed, Growing, Mature, Withering, Dead) capture that a plant responds differently to care depending on maturity (**State**). Health vitals and thresholds encode simplified horticultural rules that drive those transitions.

1.2 Staff Coordination and Task Flow

- Nursery teams span growers, floor staff, cashiers, and logistics who need shared context (shift notes, stock changes, care alerts).
- Command batching and delegation help manage repetitive jobs (watering runs, pruning, order prep).
- **Insight applied:** Care requests become `Command` objects that staff can queue, undo, or audit; specialised handlers (**Chain of Responsibility**) route tasks to the right role. A `NurseryMediator` centralises communication so staff, inventory, and customer service modules interact without tight coupling (**Mediator**).

1.3 Customer Experience and Sales Workflow

- Differentiated service (bespoke potting, gift preparation, delivery) is a competitive lever (Garden Centre Magazine, 2024).
- Checkout systems shield customers from internal stock/database complexity.
- **Insight applied:** The `SalesFacade` offers a single entry point for the GUI and demo scripts, hiding payment, stock reservation, and notification details (**Facade**). The `OrderDirector` with a `CustomOrderBuilder` composes customised orders so optional features stay modular (**Builder**).

1.4 System Scalability and Data Management

- Nurseries expand by adding greenhouse sections, experimenting with cultivars, and switching record formats as partners demand (CSV exports, plain text manifests).
- **Insight applied:** A composite greenhouse structure lets controllers treat beds and individual plants uniformly and extend the hierarchy as the site grows (**Composite**). Plant prototypes are stored in a registry so new cultivars can be cloned without revisiting construction code (**Prototype**). CSV and TXT file formats are supported through adapters, insulating inventory logic from I/O changes (**Adapter**). Observer links keep UI widgets and stock records in sync when a plant changes availability (**Observer**).

1.5 Assumptions and Definitions

- Plants are modelled with core vitals (health, water, nutrients, sunlight preference) rather than biological precision; simulation ticks abstract real-time growth.
- Customerpreferencesfocusonordercustomisationfeaturesexposedthroughthebuilder(pot, wrapping, delivery); financials remain high level.
- The nursery operates in a controlled climate so environmental factors enter via strategies and state thresholds instead of free-form physics.

1.6 References

- Department of Agriculture.(2023).*Greenhouse Management and Plant Propagation Guide.* Pretoria.
- Royal Horticultural Society. (2022). *Plant Care Best Practices.* London.
- Garden Centre Magazine. (2024). *Trends in Customer Personalisation and Retail Nurseries.*

2. System Overview

The Plant Nursery Simulator models a nursery business end-to-end:

- **Greenhouse:** Plants are organised in a **Composite** of beds and leaves. A **Greenhouse Iterator** traverses all plants in depth-first order to apply periodic growth ticks and care.

- **Plants and lifecycle:** Each plant instance holds vitals (health, water, nutrients) and a lifecycle **State** (Seed → Growing → Mature → Withering → Dead). States encapsulate per-stage behaviour; **Strategy** objects (watering, fertilising) update vitals.

- **Observation:** Plants are **Subjects** and notify **Observers** (Staff, StockItem) about care needs and availability changes.

- **Inventory and sales:** Inventory maintains `StockItem`s (some linked to `PlantInstance`). **SalesFacade** orchestrates sales, orders, inventory updates, and uses the **Builder** to assemble orders.

- **Persistence: Adapter**s (CSV/TXT) abstract I/O so Inventory remains independent of file formats.

- **Operations:** Care actions are **Commands** queued and executed by Staff; care requests route through a **Chain of Responsibility**; Customer/Staff communication goes via a **Mediator**.

3. Functional & Non-Functional Requirements

3.1 Functional Requirements (FR)

The anchors below let pattern sections link directly to these FRs.

- **FR1.** Register clonable plant prototypes by name. *(Prototype)*
- **FR2.** Create plants by cloning a registered prototype. *(Prototype)*
- **FR3.** Model plant lifecycle states and transitions during growth ticks. *(State)*
- **FR4.** Mark a plant as available for sale when market-ready. *(State)*
- **FR5.** Define interchangeable watering strategies and execute per plant. *(Strategy)*
- **FR6.** Define interchangeable fertilising strategies and execute per plant. *(Strategy)*

- **FR7.** Allow changing strategies at runtime per plant instance. *(Strategy)*
- **FR8.** Provide a file I/O abstraction for inventory. *(Adapter Target)*
- **FR9.** Load and save inventory via pluggable adapters (CSV, TXT). *(Adapter)*
- **FR10.** Organise plants in a composite greenhouse hierarchy. *(Composite)*
- **FR11.** Cascade care operations through the greenhouse hierarchy. *(Composite)*
- **FR12.** Provide an iterator to traverse plants in a bed hierarchy. *(Iterator)*
- **FR13.** Plants act as subjects that notify observers of events. *(Observer)*
- **FR14.** Staff and stock items observe plants and react to events. *(Observer)*
- **FR15.** Expose an aggregate interface for inventory iterators. *(Iterator Aggregate)*
- **FR16.** Iterate over inventory items for browsing and sales. *(Iterator)*
- **FR17.** Maintain and query inventory (add, remove, count, lookup).
- **FR18.** Encapsulate plant care actions as commands. *(Command)*
- **FR19.** Staff queue and invoke plant care commands. *(Command)*
- **FR20.** Route care requests through a chain of responsibility. *(CoR)*
- **FR21.** Each care handler processes or delegates. *(CoR)*
- **FR22.** Coordinate communication between colleagues via mediator. *(Mediator)*
- **FR23.** Customers and staff send/receive messages via mediator. *(Mediator)*
- **FR24.** Build orders using a builder and optional director. *(Builder)*
- **FR25.** Provide a unified façade for sales operations. *(Facade)*
- **FR26.** Execute periodic growth ticks across all plants. *(Controller + Iterator)*
- **FR27.** Keep stock availability in sync with plant readiness. *(Observer)*

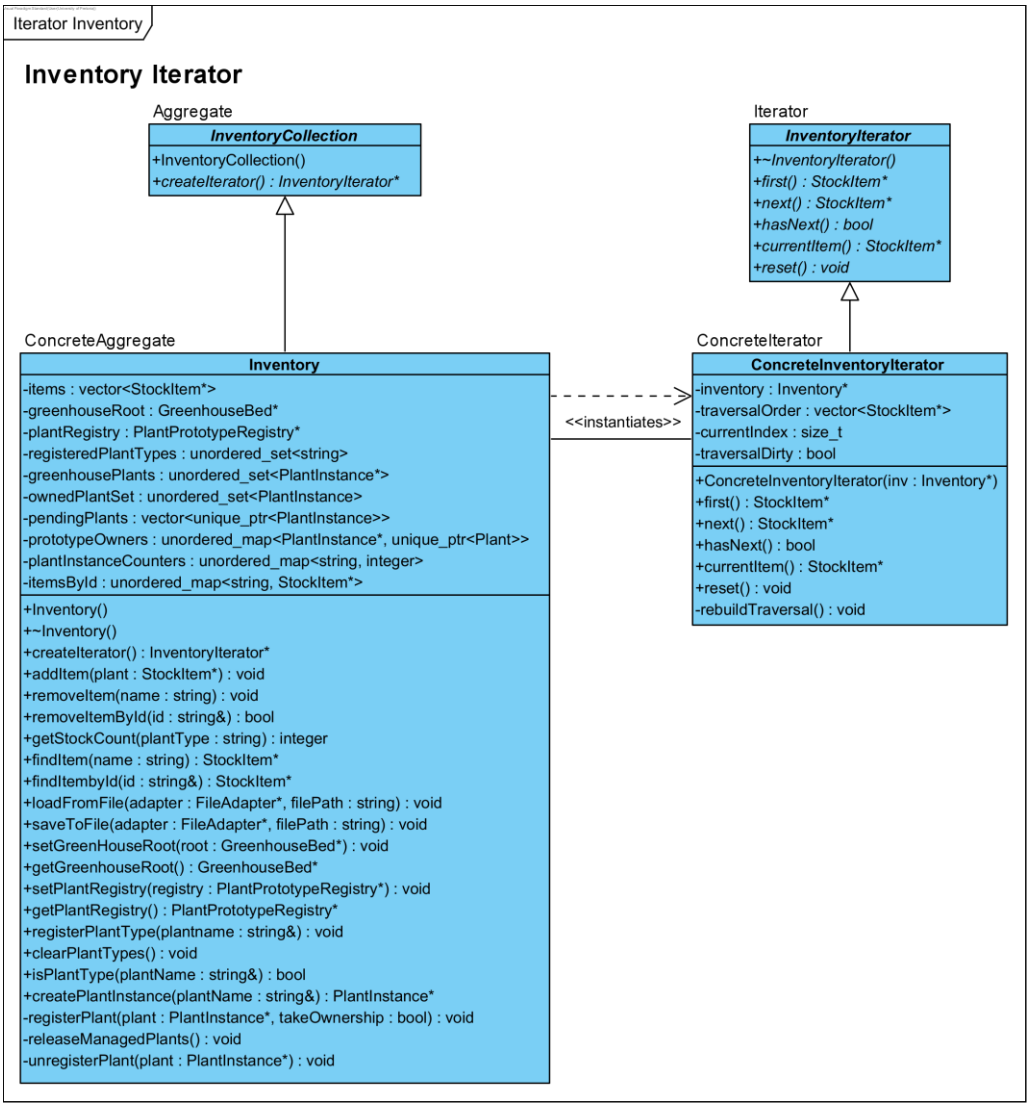
3.2 Non-Functional Requirements (NFR)

- **NFR1 (Scalability, Performance).** Traversal and growth ticks scale linearly with plant count (O(n)).
- **NFR2 (Performance).** Growth tick uses bounded constant-time updates per plant.
- **NFR3 (Reliability).** Observers never access destroyed subjects; shutdown is communicated.
- **NFR4 (Extensibility).** New strategies, states, handlers, iterators, adapters, and commands can be added without modifying client code.
- **NFR5 (Usability, Maintainability).** Sales interactions simplified through a single façade.
- **NFR6 (Portability, Maintainability).** Persistence adaptable to multiple file formats without changing inventory logic.
- **NFR7 (Documentation).** Core components documented and organised for Doxygen generation.
- **NFR8 (Testability, Maintainability).** Interfaces and separation of concerns support unit testing.

4. Design Patterns Report

Note: Each pattern lists intent, participants, implementation evidence, FR mapping, and a brief "Why not X" trade-off to satisfy rubric depth.

4.1 Iterator — Inventory



- **Intent & rationale:** Uniform traversal over inventory without exposing containers; enforce business ordering.

-**Where:**`InventoryCollection.h`,`InventoryIterator.h`, `ConcreteInventoryIterator.*`, `Inventory.*`

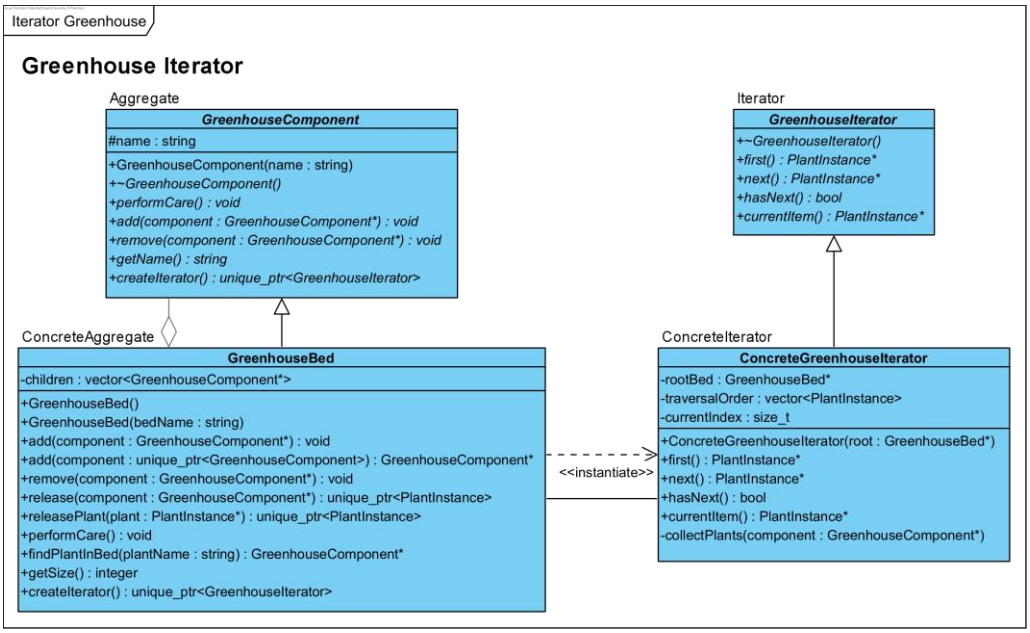
- **Participants:** `InventoryIterator` (Iterator), `ConcreteInventoryIterator` (ConcreteIterator), `InventoryCollection` (Aggregate), `Inventory` (ConcreteAggregate) - **Key interactions:** `Inventory::createIterator()` → `hasNext()/next()`

- **FRs:** FR15, FR16

- **Why not returning vectors?** Would leak representation and produce stale copies; iterator recomputes order lazily and hides storage.

- **Efficiency:** O(n) rebuild; lazy recalculation.

4.2 Iterator — Greenhouse



-**Intent& rationale:** Traverse Composite greenhouse without coupling traversal to structure.

- **Where:**`GreenhouseIterator.h`, `ConcreteGreenhouseIterator.*`,`GreenhouseComponent.h`, `GreenhouseBed.*`,`GreenhouseController.*`

-**Participants:** `GreenhouseIterator`, `ConcreteGreenhouseItera` `GreenhouseComponent`, `GreenhouseBed`

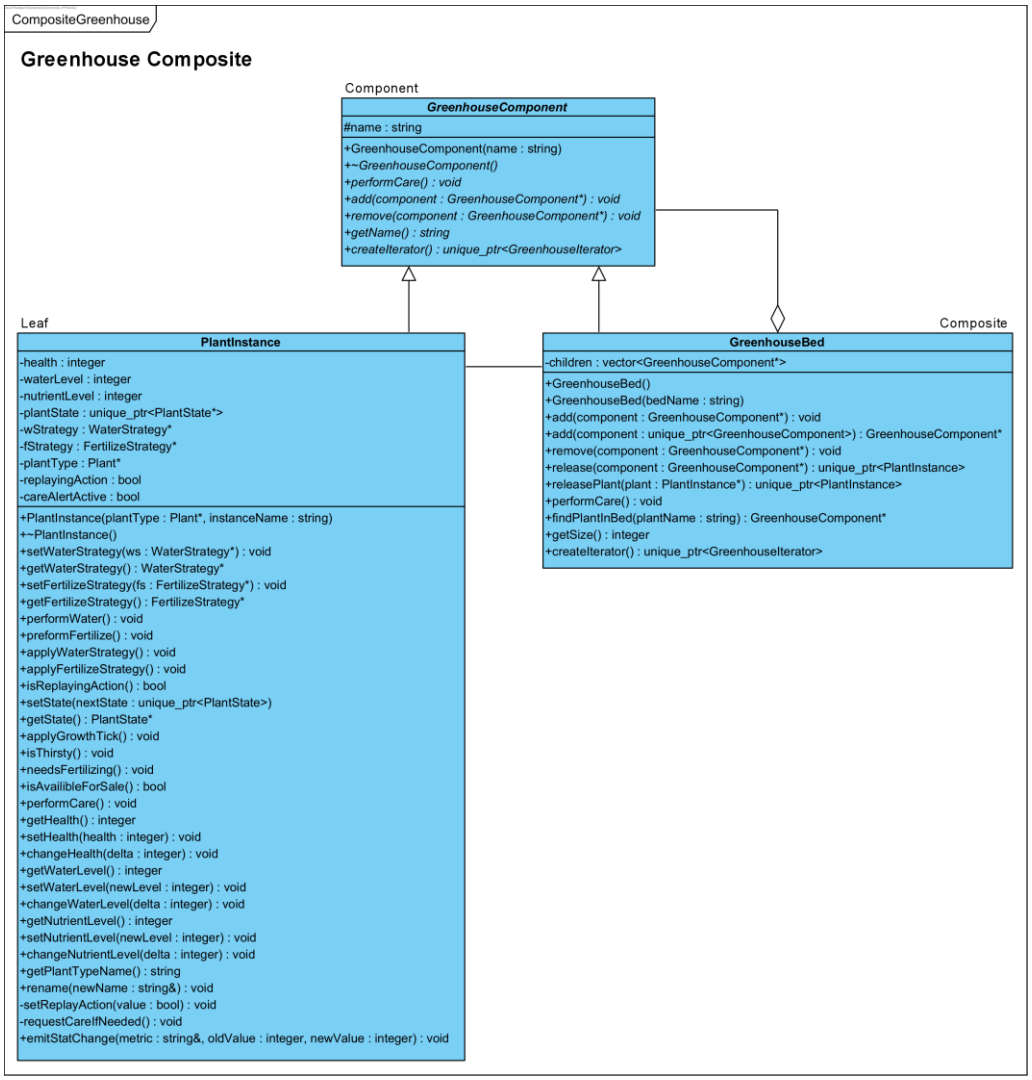
- **Key interactions:** Controller requests iterator and applies growth ticks.

- **FRs:** FR12, FR26 –

Why not recursion in controller? Couples controller to composite shape and repeats code; iterator centralises traversal logic.

- **Efficiency:** DFS collection; O(n) iteration.

4.3 Composite — Greenhouse Structure



-**Intent & rationale:** Part–whole hierarchy of beds and plants;uniform operations.

- **Where:**

`GreenhouseComponent.h`, `GreenhouseBed.*`, `PlantInstance.*`

- **Participants:** Component (`GreenhouseComponent`), Composite (`GreenhouseBed`), Leaf (`PlantInstance`)

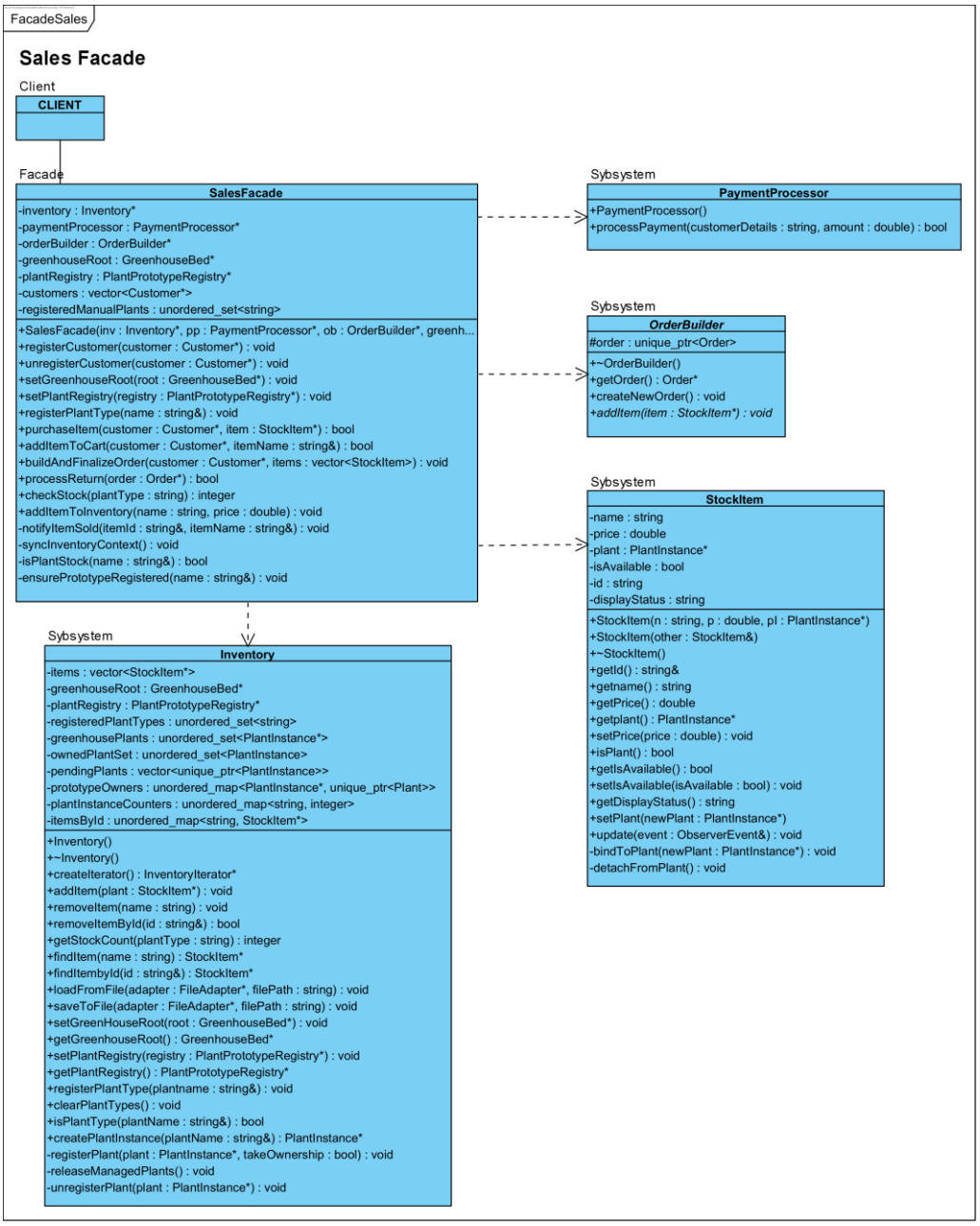
- **Key interactions:** `performCare()` cascades from composite to leaves.

- **FRs:** FR10, FR11, FR26

- **Why not flat lists?** Loses structural grouping and makes bulk operations verbose; Composite keeps client code uniform.

-**Efficiency:** Single-pass cascades; safe ownership.

4.4 Facade — Sales



-**Intent & rationale:** Single entry point for sales operations decoupling UI from subsystems.

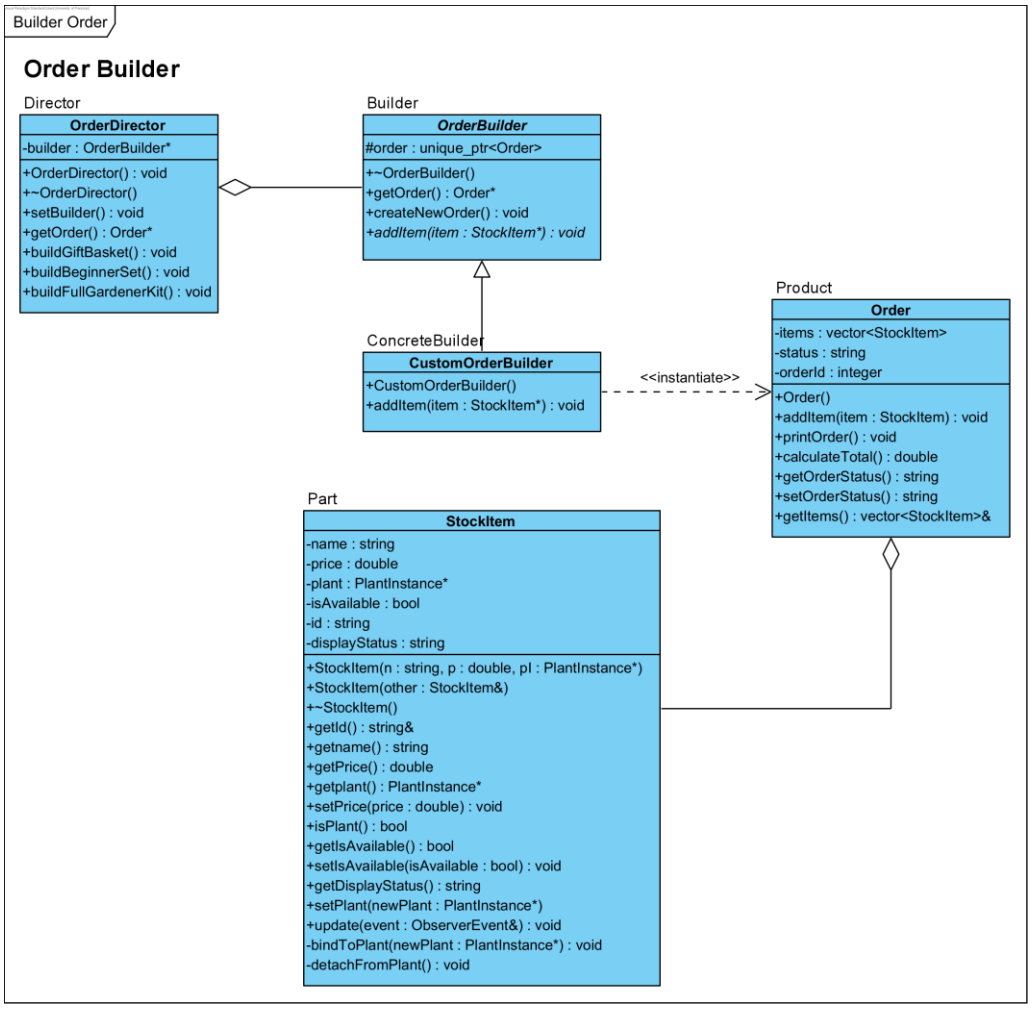
- **Where:** `SalesFacade.*` (+ `Inventory`, `PaymentProcessor`, `OrderBuilder`)

- **FRs:** FR25

- **Why not Service Locator?** Encourages hidden dependencies and global state; Facade narrows a stable API while keeping subsystems swappable.

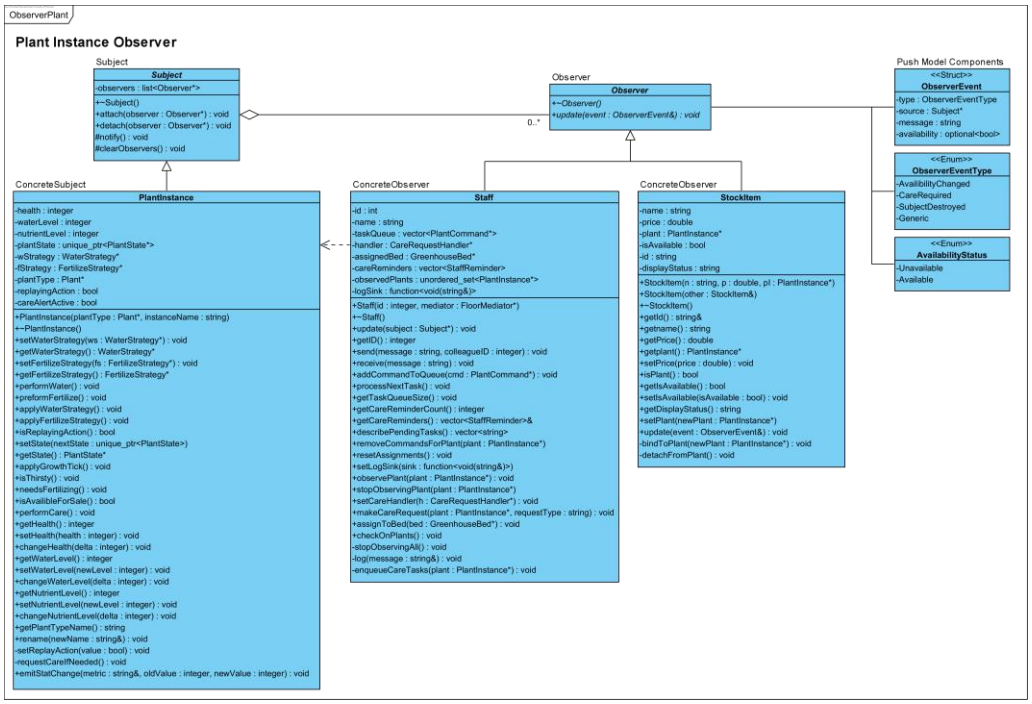
- **Efficiency:** O(1) orchestration per item.

4.5 Builder — Orders



- **Intent & rationale:** Step-by-step construction of complex orders via director presets.
- **Where:** `Order.*`, `OrderBuilder.h`, `CustomOrderBuilder.*`, `OrderDirector.*`
- **Participants:** Product (`Order`),
Builder (`OrderBuilder`), ConcreteBuilder (`CustomOrderBuilder`), Director (`OrderDirector`)
-**FRs:** FR24
- **Why not Abstract Factory?** Fixed combinations suit factories; variable, validated, stepwise composition fits Builder.
- **Efficiency:** Linear in item count.

4.6 Observer — Plants, Staff, Stock Items



- **Intent & rationale:** Decouple plant events from staff reminders and storefront availability.

- **Where:** `Observer.h`, `Subject.*`, `Staff.*`, `StockItem.*`, `PlantInstance.*`

- **Participants:** Subject (`PlantInstance`), Observer (`Staff`, `StockItem`)

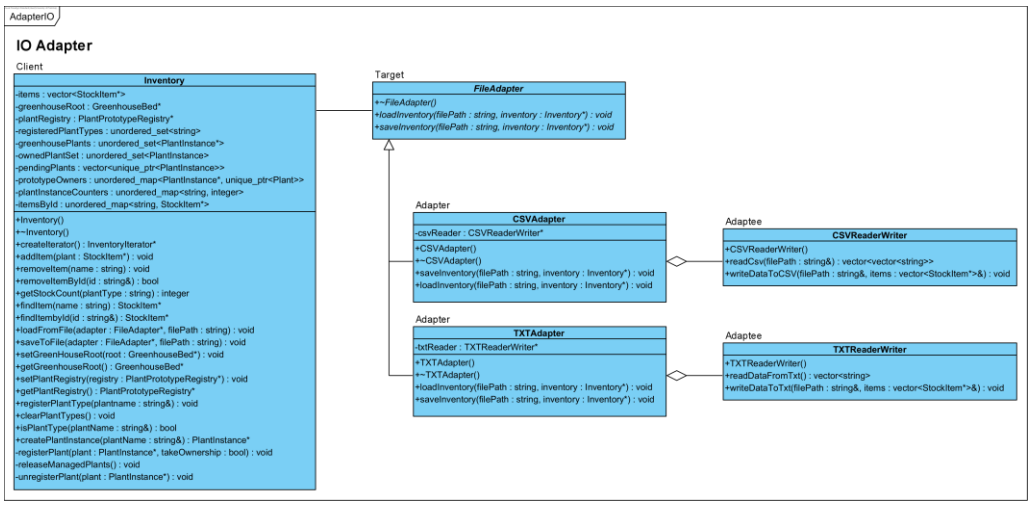
- **Key interactions:** Push-model notifications with snapshot payloads.

-**FRs:** FR13, FR14, FR27

- **Why not polling?** Expensive and stale; Observer provides timely updates without scans.

- **Efficiency:** O(k) per notify; minimal payloads.

4.7 Adapter — Inventory File I/O



- **Intent & rationale:** PluggableCSV/TXTI/Owithoutchanginginventorylogic.

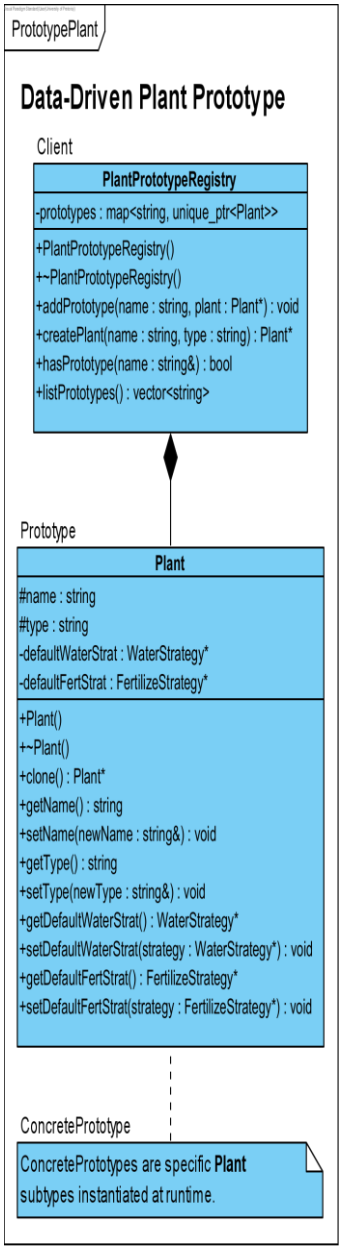
-**Where:** `FileAdapter.h`(Target), `CSVReaderWriter.*` / `TXTReaderWriter.*` (Adaptees), `CSVAdapter.*` / `TXTAdapter.*` (Adapters); `Inventory` uses `FileAdapter`.

- **FRs** FR8, FR9

- **Why not Strategy alone?** We're adapting incompatible interfaces into a common target API; classic Adapter fit.

- **Efficiency:** Single-pass encode/decode; streams via iterators.

4.8 Prototype — Plants and Registry



-Intent&rationale:

Data-driven creation of plant types by cloning prototypes.[12]

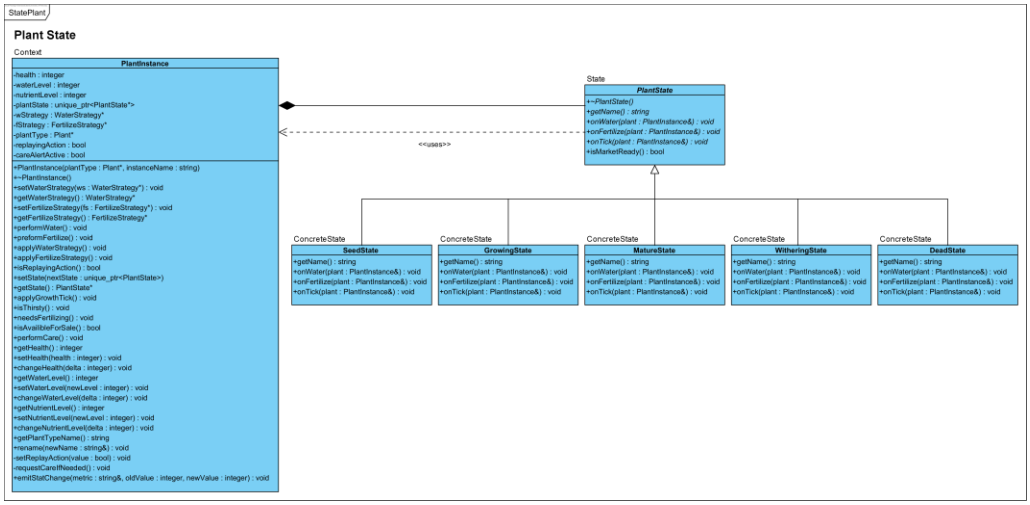
- **Where:** `Plant.* (clone()), PlantPrototypeRegistry.*`

- **FRs:** FR1, FR2 –

Why not central factory? Would require edits for every new cultivar; Prototype enables runtime registration + cloning.

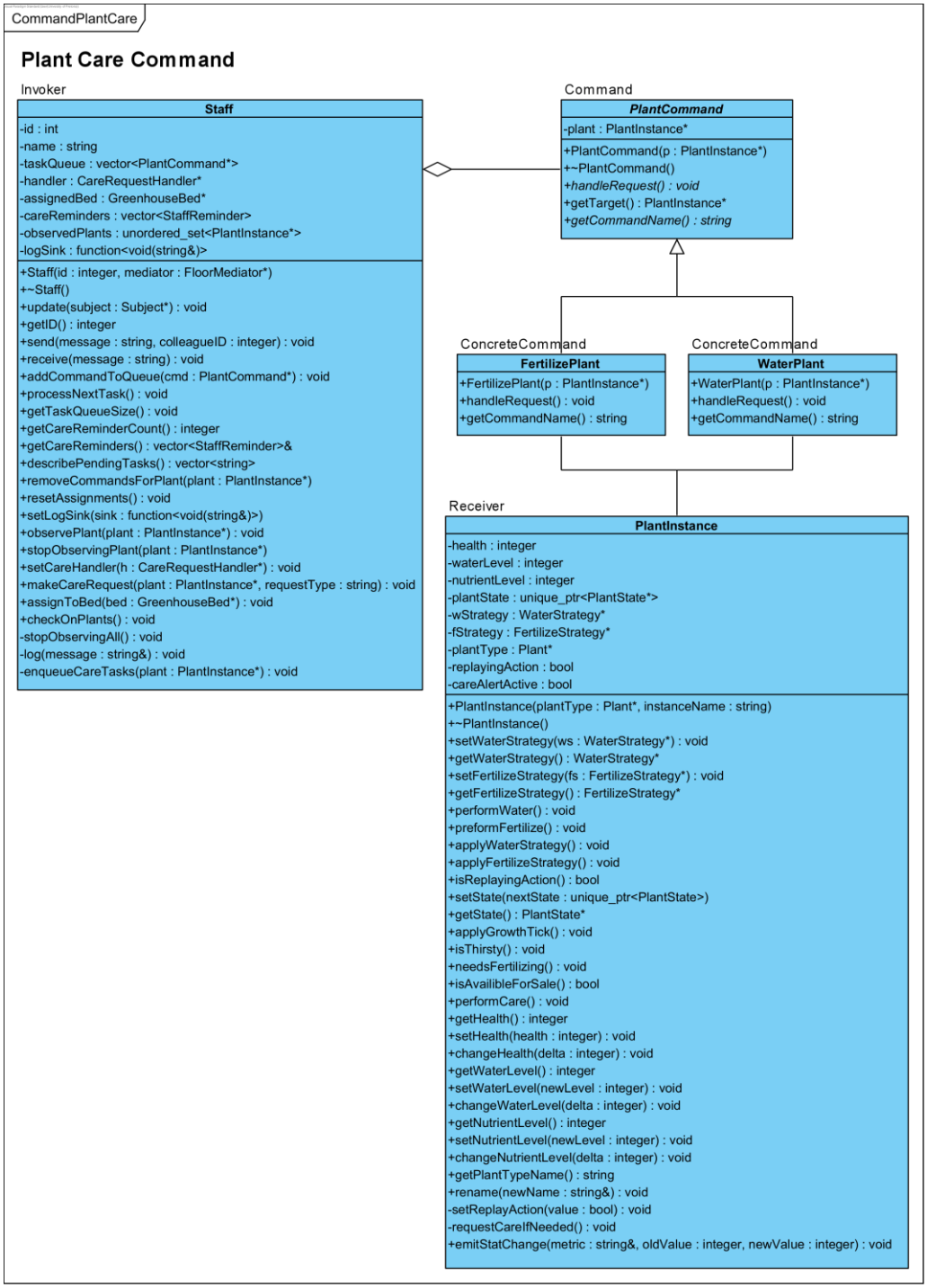
- **Efficiency:** O(1) clone; avoids reflection-heavy factories.

4.9 State — Plant Lifecycle



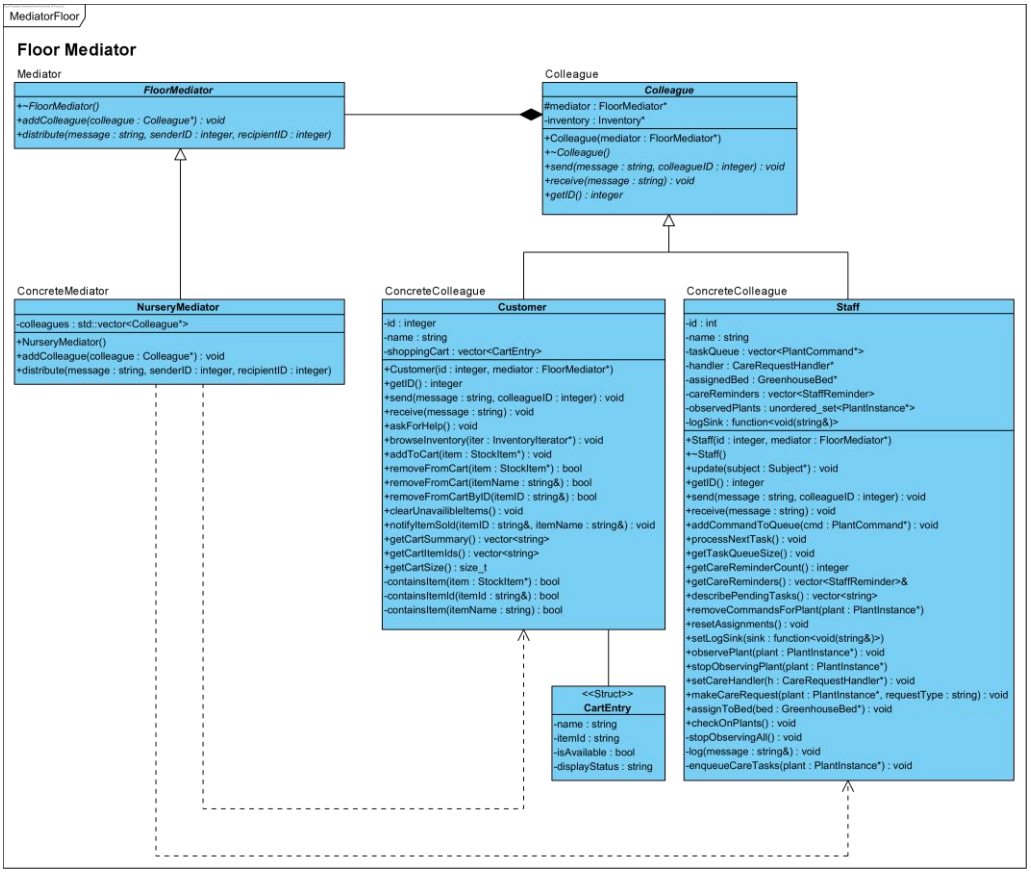
- **Intent & rationale:** Encapsulate lifecycle behaviour and transitions.
- **Where:** `PlantState.h`, concrete states (Seed/Growing/Mature/Withering/Dead), helpers (`PlantStateUtils`, `PlantStateThresholds`); `PlantInstance` delegates.
- **FRs:** FR3, FR4
- **Why not big if-else chains?** Hard to extend and reason about; State localises behaviour and transitions per stage.
- **Efficiency:** Constant-time per tick per plant.

4.10 Command — Plant Care Tasks



- **Intent & rationale:** Encapsulate care actions (water, fertilise) queued by staff.
- **Where:** `PlantCommand.h`, `WaterPlant.*`, `FertilizePlant.*`; receiver `PlantInstance`; invoker `Staff`.
- **FRs:** FR18, FR19
- **Why not direct calls/lambdas?** Commands give a typed queue, history, and future undo/redo semantics.
- **Efficiency:** O(1) enqueue/dequeue.

4.11 Mediator — Floor Coordination



- **Intent & rationale:** Centralise communication among colleagues to reduce coupling.

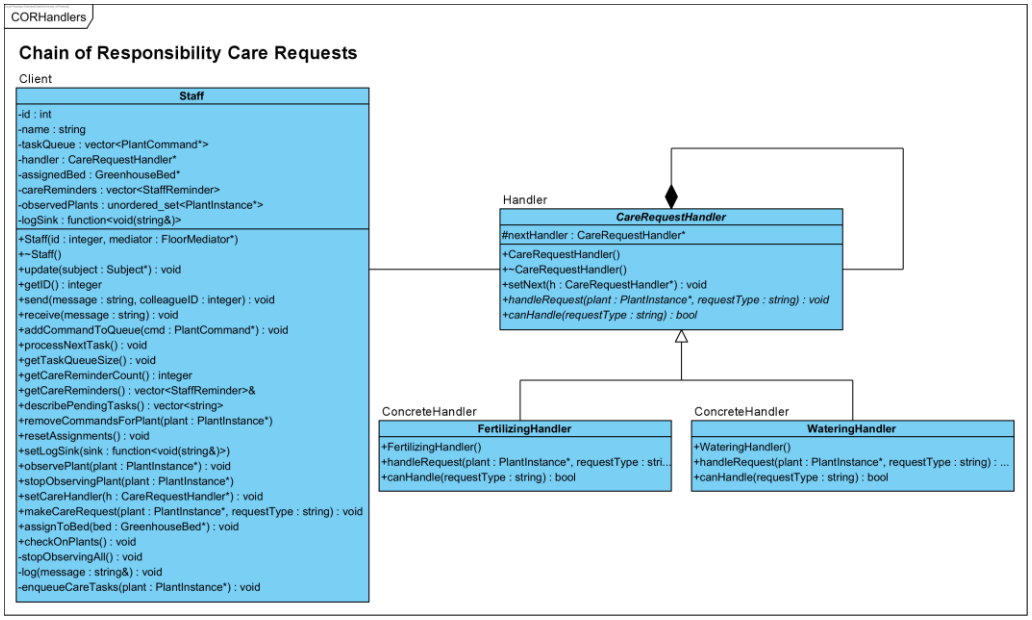
- **Where:** FloorMediator.h, NurseryMediator.*; Colleague base; Customer/Staff.

- **FRs:** FR22, FR23

- **Why not Observer for chat?** We need directed routing (unicast/broadcast) and addressing; Mediator models this explicitly

. - **Efficiency:** O(m) broadcast; O(1) targeted.

4.12 Chain of Responsibility — Care Routing



- **Intent & rationale:** Route care requests through handlers; easy extensibility.

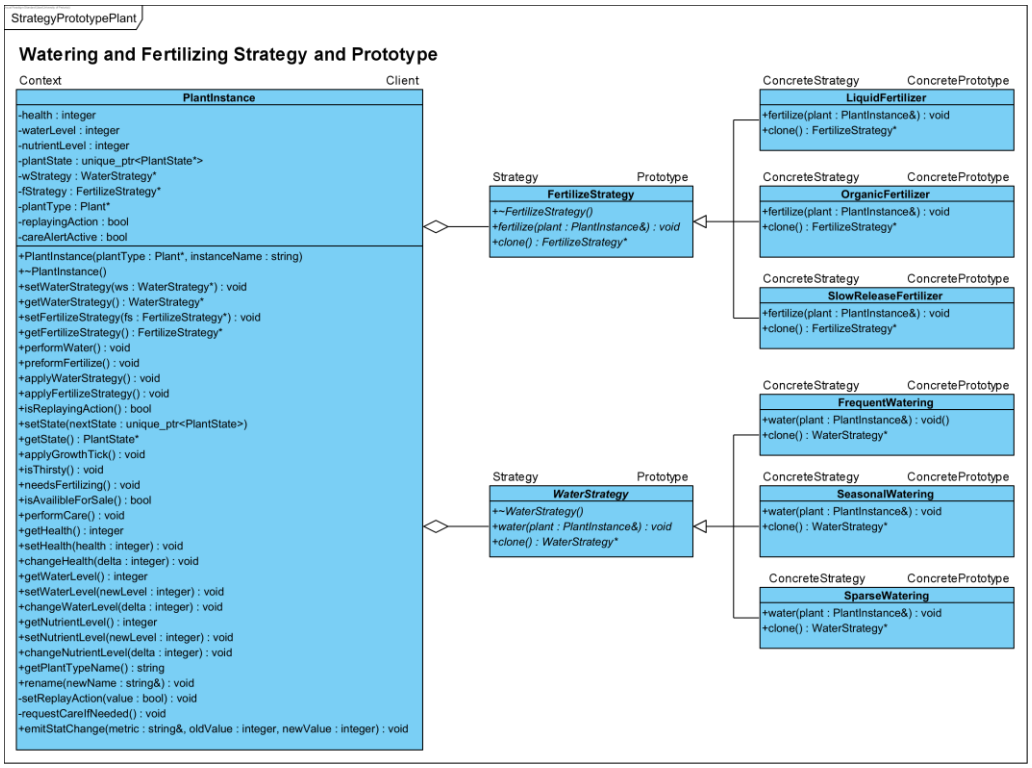
- **Where:** `CareRequestHandler.h`, `WateringHandler.*`, `FertilizingHandler.*`; wired by `Staff`.

- **FRs:** FR20, FR21

- **Why not a switch/dispatcher?** Central dispatcher violates Open/Closed; CoR adds new handlers without modifying the core.

- **Efficiency:** O(h) worst case where h is handler depth.

4.13 Strategy — Watering and Fertilising



- **Intent & rationale:** Swap care algorithms at runtime; optional prototyped presets.

- **Where:** `WaterStrategy.h`, `FertilizeStrategy.h`; concrete strategies; used by `PlantInstance`.

- **FRs:** FR5, FR6, FR7

- **Why not Command for algorithms?** Commands represent queued actions; Strategy models the algorithm used by the plant at execution time.

- **Efficiency:** Direct virtual dispatch; O(1) swaps.

5. GUI Requirements

Goal: ensure each subsystem (sales, inventory, greenhouse, staff, adapters, prototype registry, lifecycle management) is accessible and clear.

5.1 Home Dashboard

- Summary of inventory levels, recent sales, care alerts, greenhouse health.
- Notificationpanelfedbyobserverevents(`SubjectDestroyed`,`CareRequired`, `AvailabilityChanged`).

5.2 Greenhouse Visualisation

- Browse Composite hierarchy of beds and plants; inspect vitals; trigger care commands.
- Assign staff and observer subscriptions; colour-coded lifecycle badges.

5.3 Plant Lifecycle Monitor

- Visualise the state machine and thresholds; simulate growth ticks; replay care actions.

5.4 Inventory Management

- View/edit stock; add/remove items; register plant types; filter by availability and location.

5.5 File Import & Export (Adapters)

- Import TXT/CSV with preview and validation; export current snapshot with format selector.

5.6 Sales & Customer Operations (Facade)

- Carts, purchases, custom orders via builder, returns processing, quick stock checks.

5.7 Prototype Registry & Plant Templates

- Manage prototypes and default strategies; spawn plant instances.

5.8 Staff & Task Management

- Roster view, reminder inbox, command queues, CoR care requests, mediator messaging.

5.9 Care Strategy Management

- Configure strategies and assign to plants/prototypes; preview effects.

5.10 Order & Return History

- Log of orders with statuses; search/filter; export; return processing.

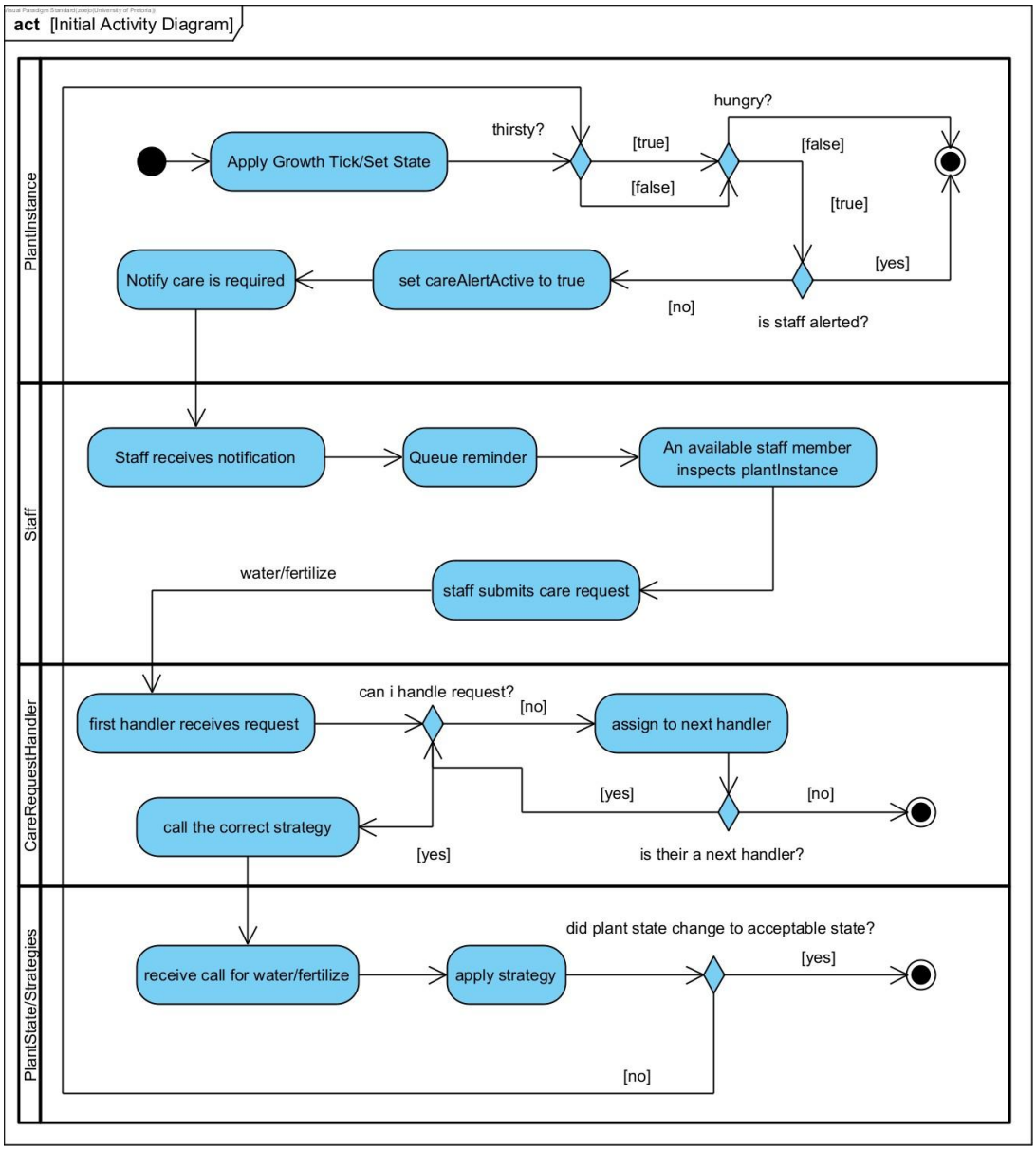
5.11 System Administration

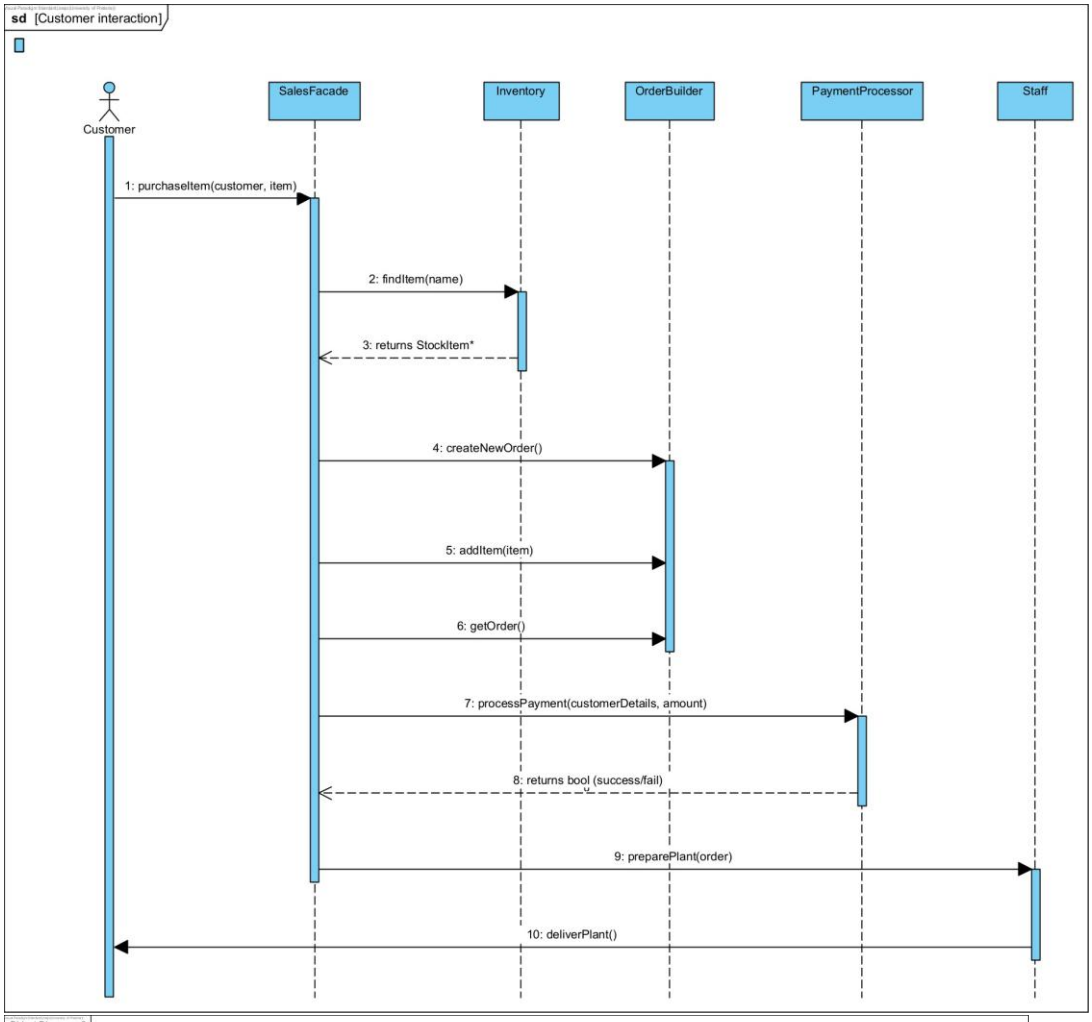
- Configure adapters, greenhouse roots, registries; diagnostics and sync tools.

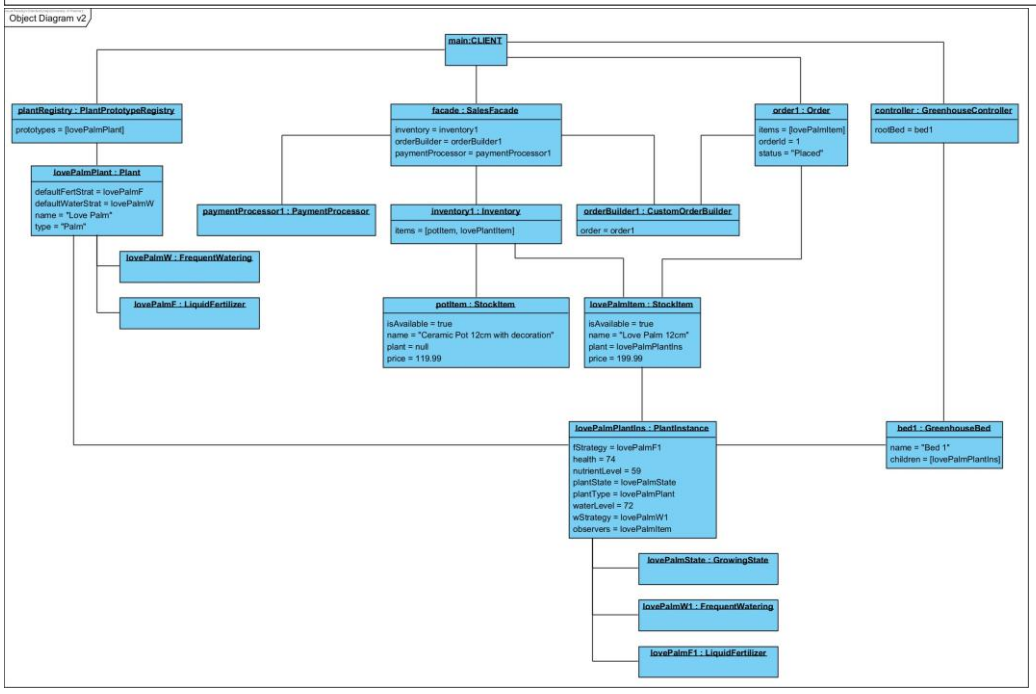
5.12 Error Handling & Notifications

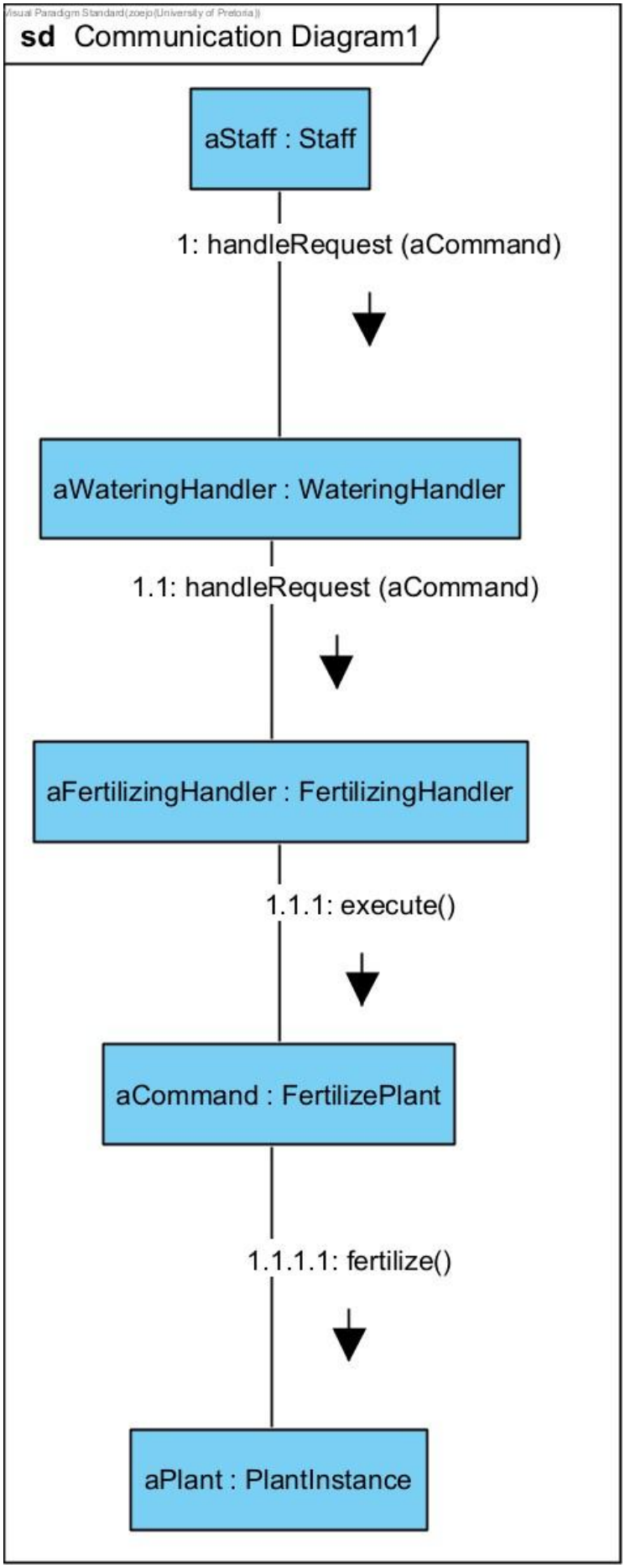
- Surface adapter errors, payment failures, missing prototypes; actionable resolutions.

Diagrams:




sd [Customer interaction]
Customer
SalesFacade
Inventory
OrderBuilder
PaymentProcessor
Staff
1: purchaseItem(customer, item)
2: findItem(name)
3: returns StockItem*
4: createNewOrder()
5: addItem(item)
6: getOrder()
7: processPayment(customerDetails, amount)
8: returns bool (success/fail)
9: preparePlant(order)
10: deliverPlant()


Object Diagram v2
main:CLIENT
plantRegistry : PlantPrototypeRegistry
prototypes = [lovePalmPlant]
facade : SalesFacade
inventory = inventory1
orderBuilder = orderBuilder1
paymentProcessor = paymentProcessor1
order1 : Order
items = [lovePalmItem]
orderId = 1
status = "Placed"
controller : GreenhouseController
rootBed = bed1
lovePalmPlant : Plant
defaultFertStrat = lovePalmF
defaultWaterStrat = lovePalmW
name = "Love Palm"
type = "Palm"
paymentProcessor1 : PaymentProcessor
inventory1 : Inventory
items = [potItem, lovePlantItem]
orderBuilder1 : CustomOrderBuilder
order = order1
lovePalmW : FrequentWatering
lovePalmF : LiquidFertilizer
potItem : StockItem
isAvailable = true
name = "Ceramic Pot 12cm with decoration"
plant = null
price = 119.99
lovePalmItem : StockItem
isAvailable = true
name = "Love Palm 12cm"
plant = lovePalmPlantIns
price = 199.99
lovePalmPlantIns : PlantInstance
fStrategy = lovePalmF1
health = 74
nutrientLevel = 59
plantState = lovePalmState
plantType = lovePalmPlant
waterLevel = 72
wStrategy = lovePalmW1
observers = lovePalmItem
bed1 : GreenhouseBed
name = "Bed 1"
children = [lovePalmPlantIns]
lovePalmState : GrowingState
lovePalmW1 : FrequentWatering
lovePalmF1 : LiquidFertilizer


Visual Paradigm Standard(zoejo(University of Pretoria))
sd Communication Diagram1
aStaff : Staff
1: handleRequest (aCommand)
aWateringHandler : WateringHandler
1.1: handleRequest (aCommand)
aFertilizingHandler : FertilizingHandler
1.1.1: execute()
aCommand : FertilizePlant
1.1.1.1: fertilize()
aPlant : PlantInstance

6. Plant Lifecycle State Machine

Each `PlantInstance` owns a polymorphic `PlantState` responsible for reacting to watering, fertilising, and tick events. Configured **Strategies** are executed through the state layer, allowing behaviour that adapts as the plant progresses. **Shared concepts**

- Unique ownership of active state via `std::unique_ptr<PlantState>`; observers receive `SubjectDestroyed` on teardown.
- Strategies invoked within state hooks; *promotion replays* run the new state exactly once with guards.
- Resource stress and recovery thresholds drive transitions; observer notifications on availability changes.

Seed

- Care boosts resources and health; tick penalises low resources.
- Transitions: → **Growing** when thresholds met; → **Dead** on critical health. **Growing**
- Care applies strategies; tick gains health if resources healthy or loses otherwise.
- Transitions: → **Withering** on stress; → **Mature** on thresholds; → **Dead** on critical health.

Mature

- Care applies strategies modestly; tick slight decline or sharper under stress.
- Transitions: → **Withering** on stress/low health; → **Dead** on critical health.

Withering

- Care provides strong recovery; tick degrades health until rescued or dead.
- Transitions: → **Dead** on critical; → **Mature/Growing** on recovery.

Dead

- No-ops; terminal.

Promotion replays

- Actions that cause a state change replay once on the new state; guards prevent double application.

Health adjustments & testing

- Values clamped 0..100; unit tests verify lifecycle and replay correctness.

7. Pattern-to-Requirement Summary

- **Prototype (Plants):** FR1, FR2
- **State (Lifecycle):** FR3, FR4
- **Strategy (Water/Fertilise):** FR5, FR6, FR7

- **Adapter (I/O):** FR8, FR9
- **Composite (Greenhouse):** FR10, FR11, FR26
- **Iterator (Greenhouse):** FR12, FR26
- **Observer (Plants/Staff/Stock):** FR13, FR14, FR27
- **Iterator (Inventory):** FR15, FR16
- **Command (Care):** FR18, FR19
- **Chain of Responsibility:** FR20, FR21
- **Mediator (Floor):** FR22, FR23
- **Builder (Orders):** FR24
- **Facade (Sales):** FR25

8. Spec Compliance & Submission Checklist

- ≥ **10 design patterns** used; covers creational, structural, behavioural.
- **Diagrams in Visual Paradigm:** Class, State, Activity, Sequence, Object, Communication (exported as PNG/JPG and embedded above).
- **Doxygen** used to generate code documentation; headers include docblocks. *HTML output at* `docs/doxygen/html/index.html` *(generated with* `doxygen Doxyfile`*)*.
- **Git & GitHub:** branching strategy documented; ≥10 commits per member; ≥3 closed PRs.
- **Automated unit tests** implemented; core flows covered (iterators, state transitions, adapters, facade).
- *(Bonus)* **GitHub Actions** for lint, test, build; *(Bonus)* **GUI** implementation.
- **Repository content:** `/src`, `/include`, `/docs` (Doxygen), `/Report` (PDF), `/data` (optional), `tests/`, `README.md`, `Makefile`, `Doxyfile`, plus ClickUp/Fitchfork submission as per spec.

9. References

- Department of Agriculture.(2023).*Greenhouse Management and Plant Propagation Guide.* Pretoria.
- Royal Horticultural Society. (2022). *Plant Care Best Practices.* London.
- Garden Centre Magazine. (2024). *Trends in Customer Personalisation and Retail Nurseries.*