Group Name: The girls + Kyle
Group Members:

Megan Azmanov - u24575292
Gabi De Gouveia - u24594084
Gabriela Berimbau - u24711013
Rachel Clifford - u24647374
Kyle McCalgan - u24648826
Kahlan Hagerman – u24601358
Sofia Finlayson - u24593240

Design Pattern Summary:

Functional Requirements

FR1	**State:** Plant lifecycle management
FR2	**Strategy:** Dynamic plant care scheduling
FR3	**Observer**, **Command:** Automatic plant monitoring & care triggers
FR4	**Mediator**, **Chain of Responsibility:** Coordinated staff operations
FR5	**Factory Method:** Factory-based plant creation
FR6	**Decorator:** Product customization
FR7	**Composite**, **Iterator:** Hierarchical composite order structure
FR8	**Prototype**, **Iterator**, **Composite:** Deep cloning of finalized orders
FR9	**Template Method:** Standardized checkout and payment flow
FR10	**Mediator**, **Composite**, **Prototype**, **Template Method:** System-wide integration and communication

Non Functional Requirements

NFR1	**Maintainability and Extensibility**
NFR2	**Reliability and Consistency**
NFR3	**Scalability and System Integration**

Note:
Behavioral: State, Strategy, Observer, Command, Mediator, Chain of Responsibility, Iterator, Template Method
Creational: Factory Method, Prototype
Structural: Decorator, Composite

Functional Requirements:

FR1: Plant Lifecycle Management
The system must track each plant's lifecycle stages (Seedling → Growing → Mature → Flowering → Dead) and allow state transitions automatically based on time and care.

Feature: Plant lifecycle tracking
Function: Manage and transition between lifecycle states
Behaviour: Each plant updates daily; transitions occur when conditions are met.
Patterns: *State*
Classes: Plant, PlantState, SeedlingState, GrowingState, MatureState, FloweringState, DeadState
Criteria:

- When a plant ages and receives sufficient care, it transitions to the next state.
- Dead plants are removed from inventory automatically.

FR2: Dynamic Plant Care System
Different plant types (succulent, flower, vegetable, etc.) require unique care routines that can change dynamically.

Feature: Dynamic care scheduling
Function: Apply plant-specific watering and care rules
Behaviour: Strategies can be swapped at runtime for new care behaviours.
Patterns: *Strategy*
Classes: CareStrategy, SucculentCareStrategy, FlowerCareStrategy, VegetableCareStrategy, CareScheduler
Criteria:

- A Rose uses FlowerCareStartegy for regular watering.
- Switching to a different strategy immediately changed behaviour.

FR3: Automatic Plant Monitoring
The system must automatically detect when a plant needs water, fertilizer, or sunlight and trigger care commands.

Feature: Plant condition observer
Function: Notify observers of low moisture, light, or nutrients
Behaviour: Observers attach/detach dynamically, executing corrective commands.
Pattern: *Observer, Command*
Classes: Plant, PlantObserver, WaterObserver, FertilizeObserver, SunlightObserver, Command, WaterPlantCommand, FertilizePlantCommand, AdjustSunlightCommand
Criteria:

- If a plants moisture <30%, WaterPlantCommand executes.
- Observers can detach without breaking the update cycle.

FR4: Coordinating Staff Operations
All communication between greenhouse, sales floor, and customers is handled via a central mediator.

Feature: Centralized coordination
Function: Route requests between staff and locations
Behaviour: Mediator passes events and requests without direct coupling.

Patterns: *Mediator, Chain of Responsibly*
Classes: NurseryMediator, NurseryCoordinator, SalesAssistant, FloorManager, NurseryOwner, Request
Criteria:

- Customer requests are routed via NurseryMediator.
- Escalations follow the chain: Assistant → Manager → Owner.

FR5: Factory-Based Plant Creation
Plants are created using factory method classes, each responsible for instantiating a specific plant type (e.g., Rose, Cactus, Aloe). This ensures consistency in setup (initial state, care strategy, and observer registration) without exposing creation logic to the client.

Feature: Type-safe plant creation
Function: Create and initialize plant objects with correct states and strategies
Behaviour: Each factory subclass overrides the buildPlant() method to return the appropriate Plant subtype.
Patterns: *Factory Method*
Classes: PlantFactory, RoseFactory, CactusFactory, AloeFactory, PotatoFactory, etc.
Criteria:

- Each Factory creates one specific subclass of Plant.
- The returned object is fully initialized (state=Seedling, care strategy assigned).
- New plant types can be added by implementing a new factory subclass without altering existing code.

FR6: Product Customization
Allow customers to enhance their plant purchases with decorations or packaging.

Feature: Dynamic customization
Function: Add decorative pot, ribbon, or gift wrap
Behaviour: Decorators wrap base plant to extend behaviour and price.
Pattern: *Decorator*
Classes: Decorator, RibbonDecorator, GiftWrapDecorator, DecorativePotDecorator, Plant
Criteria:

- Multiple decorators can stack, updating both descriptions and price.

FR7: Composite Order Structure
The system must support both individual and grouped plant orders using a composite structure.

Feature: Hierarchical orders
Function: Manage orders containing multiple plants or bundles
Behaviour: Operations apply recursively to all contained elements.
Pattern: *Composite, Iterator*
Classes: Order, ConcreteOrder, Leaf, Iterator, ConcreteIterator
Criteria:

- A ConcreteOrder can hold multiple Leaf items.
- The iterator traverses nested orders for total calculation.

FR8: Order Cloning (Prototype)
Customers can quickly reorder previous purchases through deep cloning.

Feature: Fast repeat ordering

Function: Clone entire order hierarchy for a customer
Behaviour: Deep copies all child items recursively using iterator traversal.
Pattern: *Protoype, Composite, Iterator*
Classes: AbstractFinalOrder, FinalOrder, Order, ConcreteOrder, Iterator, ConcreteIterator
Criteria:

- Cloning duplicates nested structures without shared references.
- New orders have unique IDs and timestamps.

FR9: Checkout and Payment Processing
Payments must follow a consistent process while allowing different methods (cash, credit).

Feature: Structured checkout
Function: Execute standardized payment workflow
Behaviour: Core steps fixed; payment handling varies by type.
Pattern: *Template Method*
Classes: PaymentProcessor, CashPayment, CreditCardPayment
Criteria:

- **The base class defines processTransaction() sequence.**
- **Subclasses override processPayment() only.**

FR10: System Integration and Consistency
All subsystems (Greenhouse, SalesFloor, Staff, Customer, Payment, Order) interact through a single orchestrated flow managed by the Mediator and Façade-like interfaces.

Feature: Integrated workflow
Function: Support full simulation from plant creation to sale
Behaviour: Combines patterns to maintain low coupling and modularity.
Pattern: *Mediator, Composite, Prototype, Template Method*
Criteria:

- Adding new plant or payment type requires no core changes.
- All components communicate only through defined interfaces.

Non-Functional Requirements:

NFR1: Maintainability and Extensibility
The system must be easily extendable to accommodate new plant types, care strategies, payment methods, or order decorators without altering existing code. This is achieved through modular class design and pattern-based abstraction.

Justification:

- The use of Factory Method, Strategy, Decorate, and Template Method patterns ensures each subsystem can evolve independently.
- New classes (ie. LavenderFactory, BitcoinPayement) can be added by subclassing existing abstractions.

Quality Attribute: Maintainability, Extensibility

Design Mechanisms Used:

- Factory Method, for isolating plant creation.
- Strategy, for dynamic care behaviour swapping.
- Template Method, for consistent but flexible payment workflow.
- Decorator, for stacking customizations without modifying base classes.

Criteria:

- Adding a new plant type, payment method, or care strategy complies and runs without requiring edits to existing compiled classes.
- UML and code dependencies remain one-directional (no cyclic coupling introduced.)

NFR2: Reliability and Consistency
The system must ensure reliable state transitions, consistent order management, and correct payment handling throughout all runtime interactions.

Justification:

- Each plants growth state is handled by the State pattern, ensuring valid transitions only.
- The Prototype, Composite and Iterator integration guarantees cloned orders preserve structure without data corruption.
- The Template Method ensures all payments follow a verified, consistent transaction process.

Quality Attribute: Reliability, Consistency

Design Mechanisms Used:

- State, ensures deterministic transitions between plant stages.
- Prototype, enforces deep cloning of all suborders.
- Iterator, guarantees complete traversal during cloning.
- Template Method, validates and confirms payments in a defined sequence.

Criteria:

- No invalid state transitions occur under any simulated growth sequence.
- Cloned FinalOrder objects produce identical output but separate memory references.
- Payment transactions complete atomically (either fully confirmed or fully failed).

NFR3: Scalability and System Integration
The system must support increasing numbers of plants, customers, and orders without performance degradation or architectural changes. The total number of plants handled dynamically depends on the configured size of the Greenhouse and SalesFloor**.**

Justification:

- The Mediator pattern efficiently coordinates communication between staff, customers, and facilities, preventing direct coupling.

- The Composite and Iterator patterns manage large plant collections and nested order structures based on greenhouse and sales floor capacity.
- The Factory Method and Strategy patterns decouple creation and care logic, allowing new plant types to integrate smoothly.

Quality Attribute: Scalability, Integration Efficiency, Performance

Design Mechanisms Used:

- Mediator, centralizes communication and coordination.
- Iterator, enables efficient traversal of plant and order collections.
- Composite, supports scalable management of hierarchical plant structures.
- Prototype, avoids redundant recalculations by reusing cloned structures.

Criteria:

- System performance scales linearly with greenhouse and sales floor capacity.
- Adding new plant types or staff roles requires no redesign of mediator or composite logic.
- Traversal and cloning operations maintain consistent accuracy and response time.
- Maximum plant count is determined by the dimensions (rows × columns) of the SalesFloor and Greenhouse.