COS 214 Project Report

5.4.1:

For this project we decided to do some pretty substantial research into how we could structure a coded nursery environment/setting based on how real nurseries work. Most obviously, we researched plants we could use that represented our two parts of the Strategy Design Pattern - WaterStrategy and SunStrategy. For each, we implemented certain plants as requiring low, medium and high water and sun requirements based on their real features[1] [2] [3]. For increased realism we used sources to finalise the selection of plants we resulted in using, based on how much water or sun they need. For example, basil requires a lot of water and a lot of sunlight in the natural world, which translated to it being assigned a waterPreference of high and a sunPreference of high. More of our code that attempted to emulate a real nursery was our use of the Composite Design Pattern to create the waterPlant() and exposeToSunlight() functions that would, through the use of Composite, allow us to water all plants or expose all the plants in that one component to sunlight. We came up with this as an idea both to make the task easier and more streamlined but also to further replicate how a natural environment and a real nursery would work - where all plants are exposed to sun at the same time (but perhaps to differing degrees). Also, many modern nurseries use sprinkler systems to water plants at the same time of the day, which makes the management of the nursery easier in real life too[4]. From this point onwards we created a system around this nursery structure that allowed us to cater to customers with employees who worked in the nursery with a

---

[1] https://plantisima.com/incredible-plants-that-thrive-with-minimal-care-and-watering/

[2] https://completegardening.com/11-drought-tolerant-plants-that-actually-hate-sun-and-11-that-crave-it/

[3] https://www.gardendesign.com/shade/plants.html

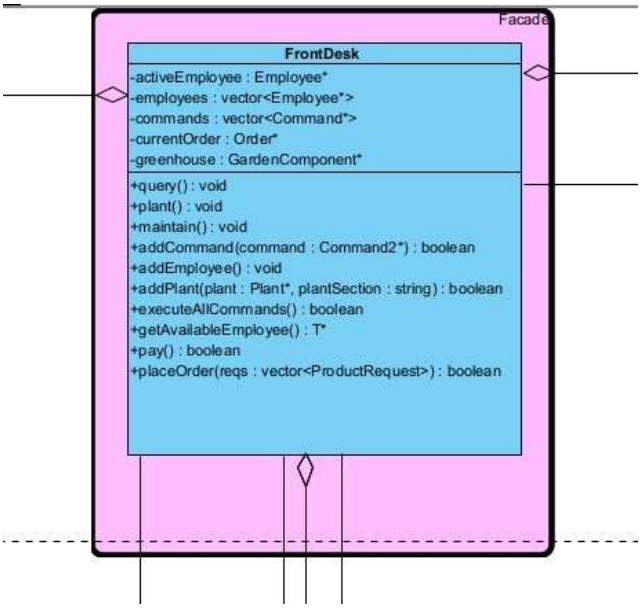
[4] https://livetoplant.com/nursery-irrigation-techniques-for-optimal-growth/

nursery environment that simulated real-life plant growth (and different plant ages) and functions that came with it, death, selling and allowing these plants to be placed in an order for customers and so on - in a manner that we felt accurately represented how a real nursery would work.

5.4.2 and 5.4.3:

Façade

The system shall provide a Facade interface through which all customer interactions occur, allowing the customer to place orders, view plant information, and receive notifications without directly accessing subsystem classes.

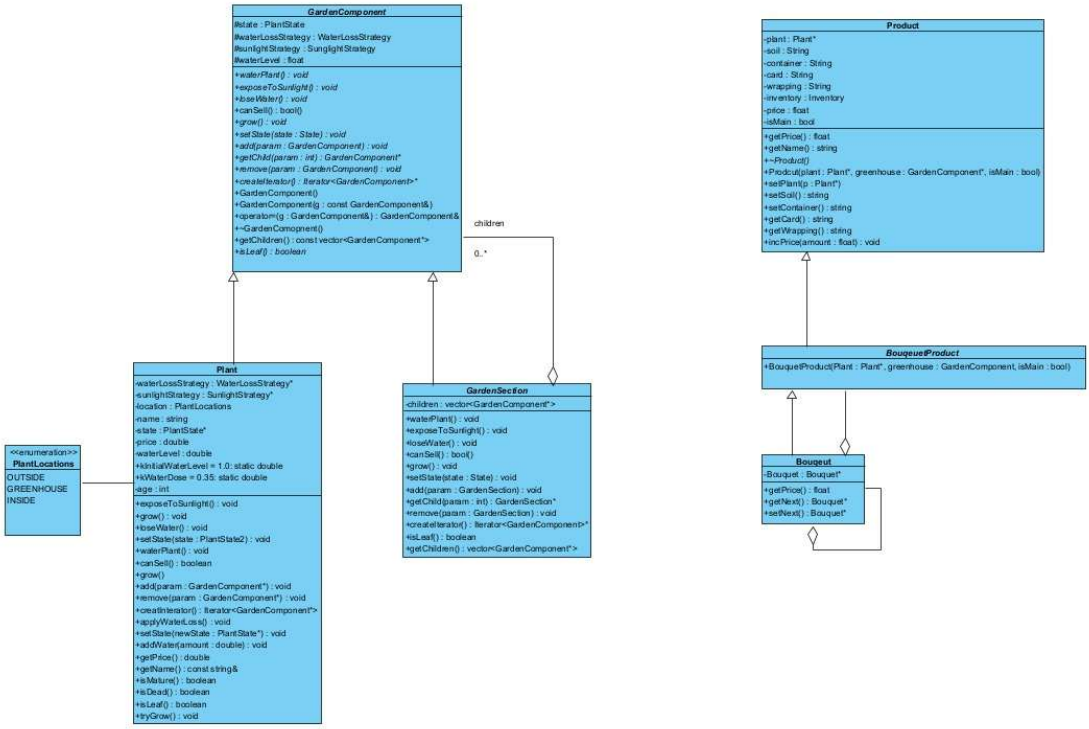


Command

The system shall encapsulate all major actions (such as placing orders, or updating inventory) as Command objects that can be queued, executed, or cancelled, enabling decoupled communication between the user interface and backend components.

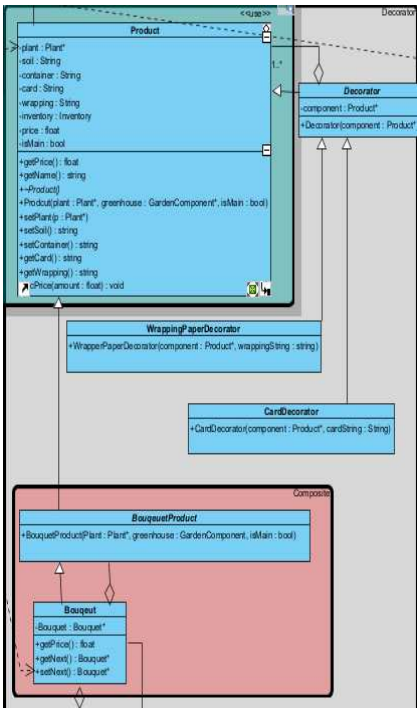
Composite

The system shall represent the garden or greenhouse hierarchy using a Composite structure, where individual plants act as Leaf components and GardenSections act as Composite nodes, enabling uniform operations (e.g., watering, growth checks) to be applied recursively across all levels.



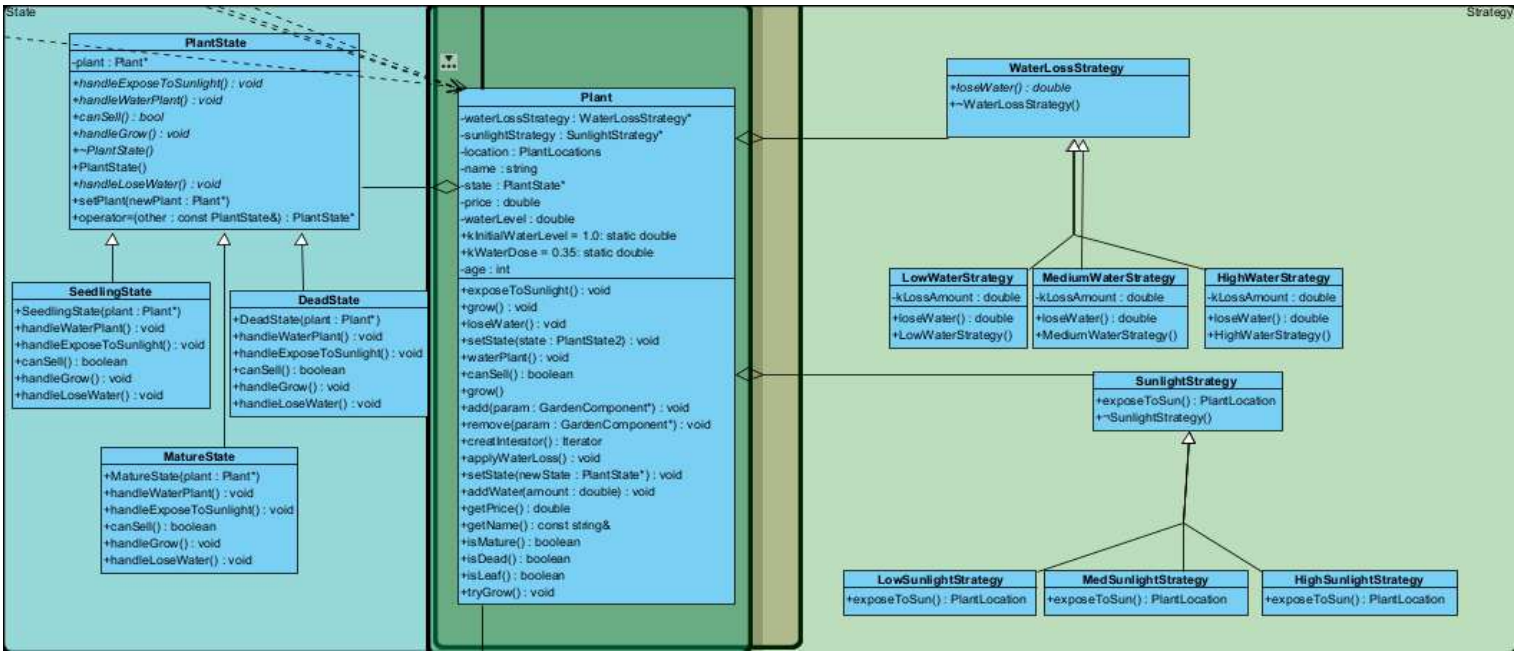
Decorator

The system uses decorator to allow extra/optional functionality to be added such as options for wrapping and a card. The decorator Design Pattern was chosen here for how it allowed optional, additional decorations/functionality to be added to a product whereas Composite was implemented in a manner that forced our products to be placed inside a bouquet if they were set to be bouquet products (where as the basic products we implemented could be stand alone and further added too with Decorator).


Decorator
«type»
Product
-plant : Plant*
-soil : String
-container : String
-card : String
-wrapping : String
-inventory : Inventory
-price : float
-isMain : bool
+getPrice() : float
+getName() : string
+~Product()
+Product(plant : Plant*, greenhouse : GardenComponent*, isMain : bool)
+setPlant(p : Plant*)
+setSoil() : string
+setContainer() : string
+getCard() : string
+getWrapping() : string
cPrice(amount : float) : void
1..*
Decorator
-component : Product*
+Decorator(component : Product*
WrappingPaperDecorator
+WrapperPaperDecorator(component : Product*, wrappingString : string)
CardDecorator
+CardDecorator(component : Product*, cardString : String)
Composite
BouqeuetProduct
+BouquetProduct(Plant : Plant*, greenhouse : GardenComponent, isMain : bool)
Bouqeut
-Bouquet : Bouquet*
+getPrice() : float
+getNext() : Bouquet*
+setNext() : Bouquet*

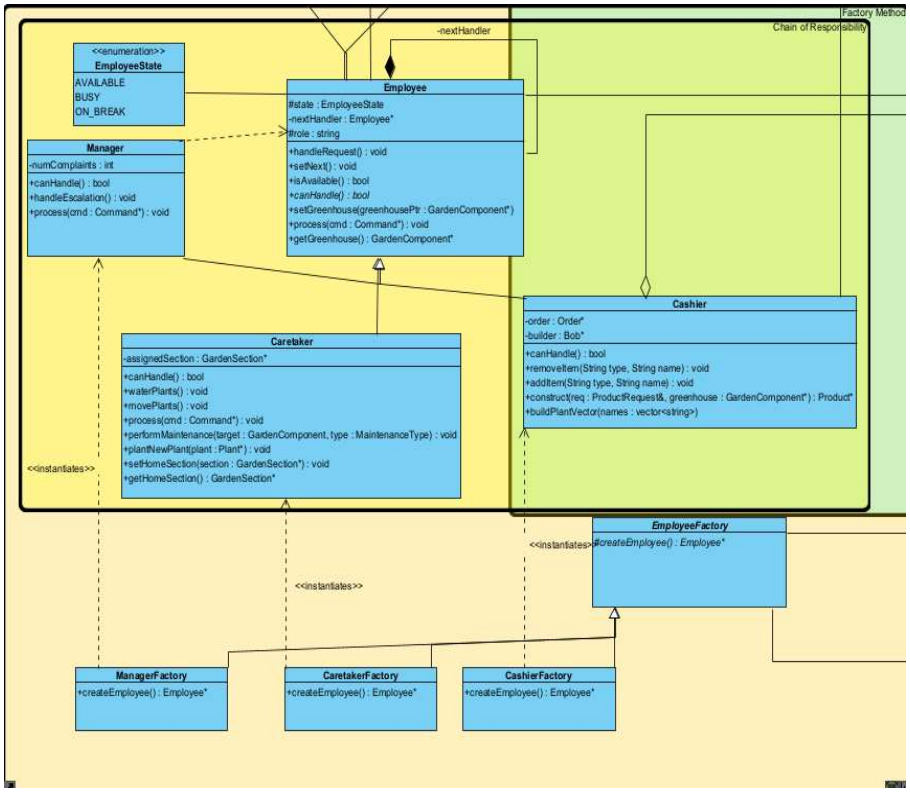
State and Strategy

Each plant shall maintain an internal State (Seedling, Mature, Dead), with transitions triggered by environmental conditions such as water and sunlight, allowing dynamic behaviour based on the plant's lifecycle stage. The system shall determine a plant's water loss rate and sunlight requirement using interchangeable Strategy objects, supporting configurable environmental behaviours (Low, Medium, or High water/sunlight dependency) for different plant types.



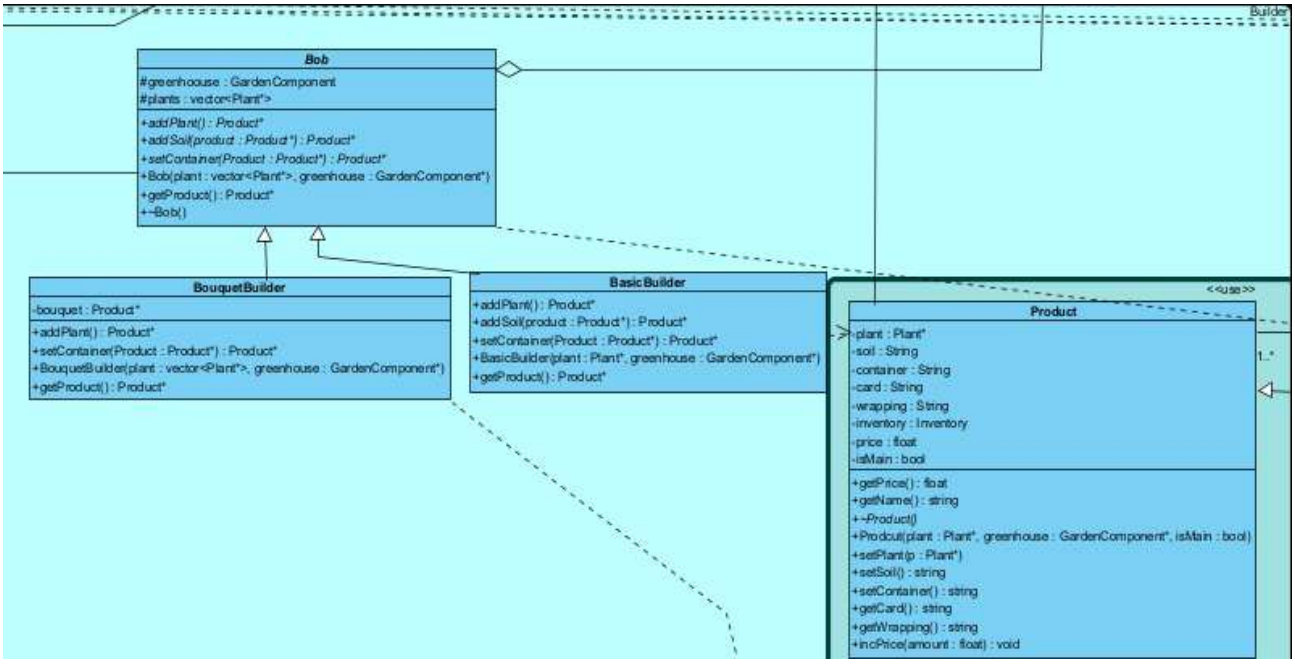
Factory and Chain of Responsibility

The system shall use a Factory component to create Employee objects (e.g., Caretaker, Cashier, Manager) as needed, ensuring centralised and consistent instantiation of staff roles. The system shall process customer service or order-related requests through a Chain of Responsibility, passing each request along a hierarchy of employees (Caretaker → Cashier → Manager) until it is handled by the appropriate authority.



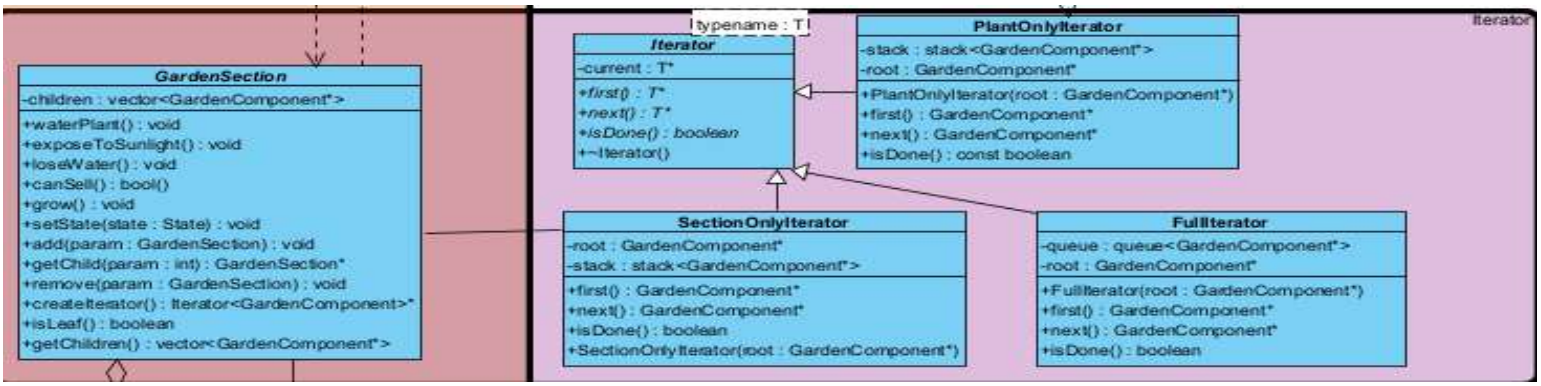
Builder

The system shall construct Product objects (e.g., basic potted plant, fancy wrapped plant, or bouquet) through the Builder pattern, enabling step-by-step creation of complex products with optional components such as soil, pots, wrappers, and cards.



Iterator

The system shall implement an Iterator to traverse the Composite garden structure, allowing ordered access to all plants and sections without exposing the internal tree representation.



Non-functional Requirements

Usability

1. The system shall provide a user-friendly and intuitive interface through the Facade, allowing customers to place orders and track their requests in no more than three clicks from the home screen.

Extensibility

2. The system shall support the addition of new plant types, employee roles, product builders, and greenhouse sections without requiring major structural changes to the existing codebase. The use of design patterns (Composite, Strategy, Factory, Builder) must allow future expansion with minimal refactoring.

Maintainability

3. The system shall be modular and easy to update, with clear separation of responsibilities across design patterns (e.g., Builders for product creation, Commands for actions). New features should be implementable by modifying or extending isolated components rather than altering core system logic.

Assumptions

- There’s a maximum amount of plants that the simulation can work with is 2000 (100 x 20 different plants).
- Plants added at beginning of sim are adults
- The maximum amount of days a system can run is 14 days; system behaviour after 14 days becomes unpredictable
- Assuming there is one cashier and manager: small business
- Inventory items (soil, containers, wrappers, cards) are always restocked at simulation start; no restocking happens mid-simulation.
- All employees start as *available* at system start.
- The order processing time is instantaneous (no waiting queues modeled).
- Employees can handle only one command or request at a time—blocking behaviour.
- Employees do not fatigue or change availability automatically; only commands affect state.
- Customer requests reference real plant names that exist in either:
1. The greenhouse *or*
2. The advice catalogue.
- Customers submit all product requests for an order at once (no partial orders).
- Plants removed for building products are permanently removed from the greenhouse composite.
- The chain of responsibility always starts with the *FrontDesk* and proceeds linearly (FrontDesk → Cashier → Manager → Caretaker, etc.).
- Only one chain is active at a time; chains do not overlap.
- If no employee can handle the request, the command is discarded with an error message.
- A “day” in the simulation is an atomic update: all plants age, lose water, etc., simultaneously.
- Weather or external conditions (humidity, temperature) are not simulated—only water/sunlight strategies matter.

Business Logic Assumptions

- All orders must be fully paid before being marked as completed.
- Customers are assumed honest; no fraud or double-pay scenarios.
- There are no refunds or returns in this simulation.