Greenhouse Simulation Final Project Report

Team Members

Name	Student Number
Hayley B Booysen	24868346
Bandile Mnyandu	24676394
Karabo Nkomo	24865169
Njabulo S Mathonsi	24676412
Marchant J Grootboom	21549232

---

1. Introduction

The Greenhouse Simulation is a sophisticated digital ecosystem that models the complete operational lifecycle of a commercial nursery. This C++-based system simulates real-world greenhouse dynamics, including plant propagation, growth management, staff operations, and customer sales interactions. The primary objective was to architect a scalable, maintainable, and extensible simulation by systematically applying object-oriented design patterns to manage inherent complexity, enhance flexibility, and promote robust code reuse.

The Justice League team focused specifically on the Nursery subsystem, with an emphasis on plant lifecycle management and propagation mechanics. This report provides a comprehensive overview of our research, detailed design rationale, implementation specifics as evidenced by the codebase, and a thorough analysis of the integrated system.

---

2. Research and Background

Our design philosophy was informed by both software engineering best practices and real-world nursery operational workflows. The system architecture reflects a modular approach where complex interactions are decomposed into manageable, loosely-coupled components.

2.1 Research Insights

- Encapsulation and Modularity: Enabled independent modification of plant care algorithms and staff behaviors, as demonstrated by the separate CareStrategy and Staff class hierarchies.

- Loose Coupling: Facilitated seamless integration of new plant types (via the Prototype pattern) and staff roles (via the Factory pattern) without necessitating changes to existing core logic.
- Extensibility: Provided a foundation for straightforward system enhancements, such as introducing new decorative elements (Decorator pattern) or specialized care routines (Command pattern).

2.2 Assumptions

The assumptions and definitions of our system will be defined below

- Zones maintain optimal conditions in terms of plant care so the plant grows year round and is not able to wilt within a zone.
- All the plants in a singular zone all grow at the same rate due to the fact that they have the same care strategy. Thus they cannot wither within their zone and can be harvested all at the same time.
- Customers will always have the correct amount needed to pay for their orders and transactions. No declines or reversal of any transactions.
- All handlers are assumed to be available and responsive when a request is passed along the chain.
- The Greenhouse system is assumed to always be operational.
- Staff members are always available, competent and never make mistakes or miss tasks.
- All users and staff are trusted; there are no unauthorized access attempts or data tampering.
- Each zone already has plants in its seedling state when the system starts

2.3 Literature Sources

Our system design drew inspiration from software engineering best practices and real-world nursery operations.
Key References:

- Gamma, Helm, Johnson, & Vlissides (1994) – Design Patterns: Elements of Reusable Object-Oriented Software

- Pressman & Maxim (2020) – Software Engineering: A Practitioner's Approach

- Fowler (2003) – Patterns of Enterprise Application Architecture

- SANBI (2023) – Nursery Management and Plant Propagation

---

3. Functional Requirements

3.1 Greenhouse / Plant System

ID	Requirement	Pattern Used

FR1	Allow propagation of plants through cloning.	Prototype
FR2	Simulate plant growth stages (Seedling → Growing → Mature → Withered).	State
FR3	Apply different care strategies for different types of plants (lowCare, mediumCare, highCare).	Strategy
FR4	Build plants dynamically with custom attributes.	Builder
FR5	Maintain a centralized inventory of plants and their seeds and keeps track of its states.	Singleton + Observer + State

3.2 Customer and Sales

ID	Requirement	Pattern Used
FR6	Enable communication between customers and staff.	Mediator
FR7	Allow customers to customize plants (pot, wrapping).	Decorator
FR8	Process purchases and record sales.	Command
FR9	Delegate delivery and request handling among staff.	Chain of Responsibility

3.3 Staff and Management

ID	Requirement	Pattern Used
FR10	Notify staff of plant changes (e.g., maturity).	Observer + Mediator
FR11	Support various staff roles (customer staff, care staff).	Factory
FR12	Enable staff to iterate through plant and seed collections.	Iterator

3.4 System Operations

ID	Requirement	Pattern Used
FR13	Manage zones containing groups of plants.	Composite
FR14	Ensure a single instance of plant inventory.	Singleton
FR15	Clone existing plants for propagation.	Prototype

4. Non-Functional Requirements

ID	Requirement	Description
NFR1	Scalability	System supports adding new plants, roles, and care types easily.
NFR2	Reliability	Maintains consistency in plant states and inventory updates.
NFR3	Performance	Executes all major actions in under one second.
NFR4	Usability	Provides an intuitive text-based interface for staff and customers.
NFR5	Modularity	Each subsystem can operate and be tested independently.

---

5. Design Patterns and Justification

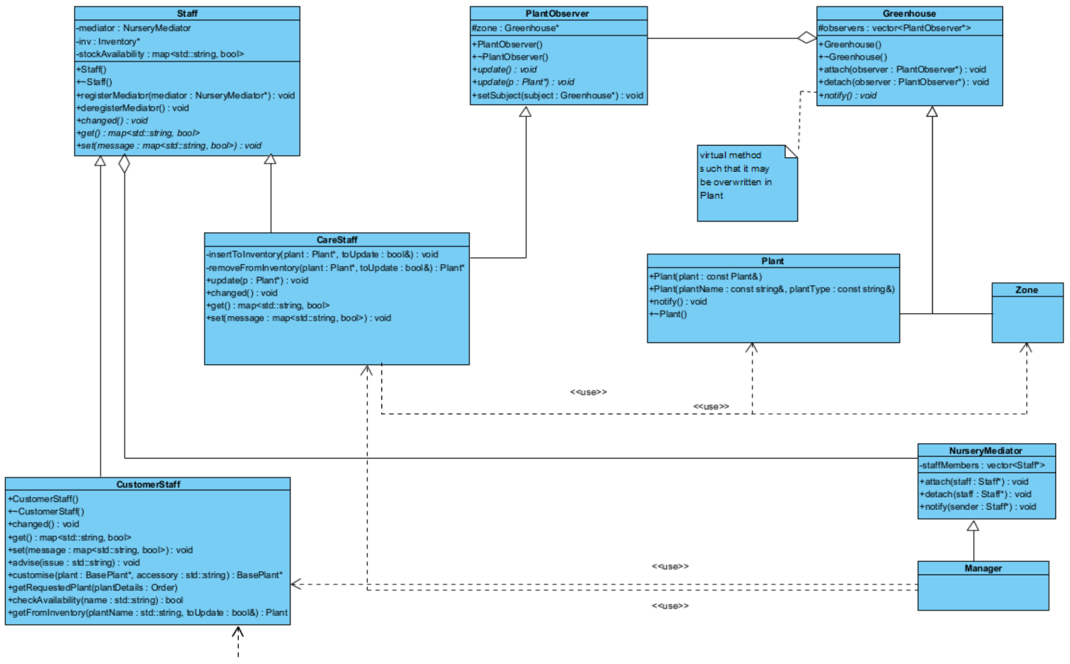
Pattern	Purpose	System Application
State (Plant Status)	Manage plant lifecycle changes	Seedling → Growing → Mature → Withered
State (Plant State)	Manage plant status in inventory	InStorage → Sold Returned → InStorage
Strategy	Allows interchangeable plant care algorithms. A Zone can be assigned a care strategy (e.g., HighCare), and all plants within it are cared for uniformly.	HighCare, MediumCare and LowCare
Observer	Notifies subscribed staff members when a plant's state changes (e.g., becomes mature and ready for sale). This decouples the plant from the staff.	Staff observers respond to maturity events
Mediator	Centralizes complex communication between Customer, Staff, and Inventory objects, preventing direct references and reducing dependencies.	Central hub for Customer–Staff–Inventory interactions
Builder	Constructs complex Plant objects step-by-step. PotPlantBuilder creates a potted plant, while WrapPlantBuilder adds	Used for creating plants with optional features

	wrapping, allowing for different representations.	
Chain of Responsibility	Delegate tasks through hierarchy	NormalStaff → Manager → SeniorManager
Iterator	Traverse lists of plants and seeds efficiently	
Singleton	Enforce single global instance	Shared Inventory object
Composite	Manage individual plants and zones uniformly	Each Zone acts as a composite container
Prototype	Clone plants for duplication or propagation	clone() used to copy existing plants
Decorator	Add optional plant customizations	PotDecorator, WrapDecorator, RibbonDecorator
Command (Care)	Encapsulate executable actions to care for plants	WaterPlant Command, FertilisePlant Command
Command (Customer)	Allow customers to be able to execute commands to customer staff	BuyPlant Command, RequestHelp Command, CustomisePlant Command
Factory	Create objects dynamically	StaffFactory, CareFactory

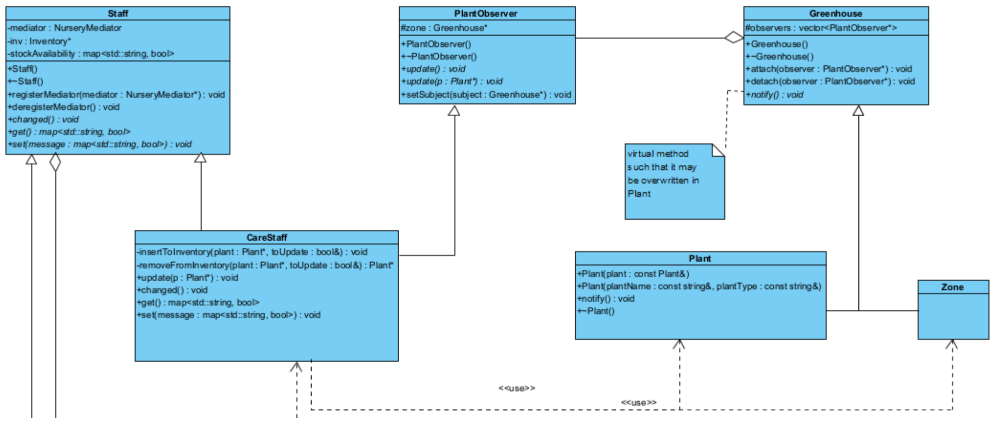
6. UML and System Design

6.1 Partial Class Diagrams

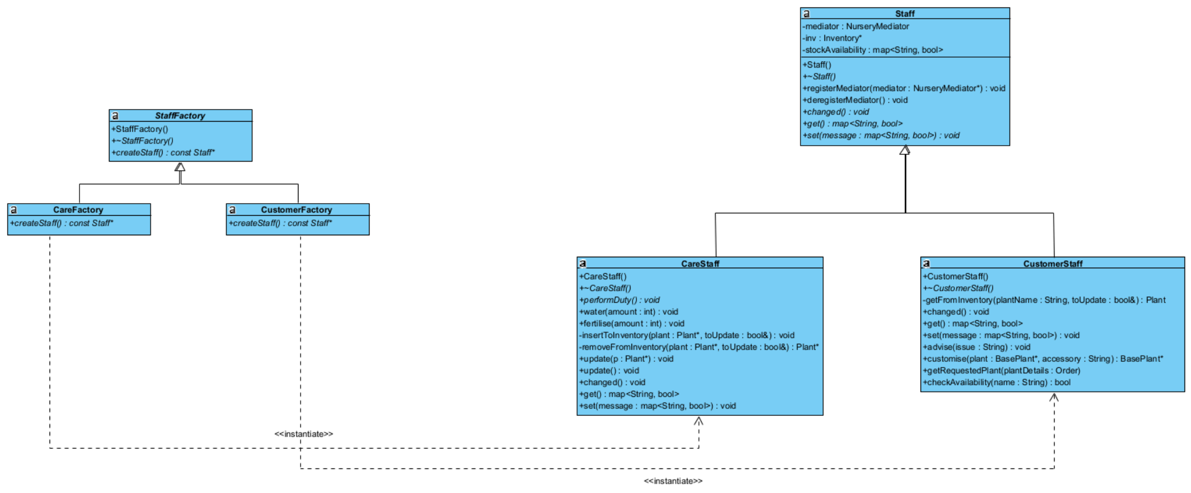
Mediator


Staff
-mediator : NurseryMediator
-inv : Inventory*
-stockAvailability : map<std::string, bool>
+Staff()
+~Staff()
+registerMediator(mediator : NurseryMediator*) : void
+deregisterMediator() : void
+changed() : void
+get() : map<std::string, bool>
+set(message : map<std::string, bool>) : void
PlantObserver
#zone : Greenhouse*
+PlantObserver()
+~PlantObserver()
+update() : void
+update(p : Plant*) : void
+setSubject(subject : Greenhouse*) : void
Greenhouse
#observers : vector<PlantObserver*>
+Greenhouse()
+~Greenhouse()
+attach(observer : PlantObserver*) : void
+detach(observer : PlantObserver*) : void
+notify() : void
virtual method such that it may be overwritten in Plant
CareStaff
-insertToInventory(plant : Plant*, toUpdate : bool&) : void
-removeFromInventory(plant : Plant*, toUpdate : bool&) : Plant*
+update(p : Plant*) : void
+changed() : void
+get() : map<std::string, bool>
+set(message : map<std::string, bool>) : void
Plant
+Plant(plant : const Plant&)
+Plant(plantName : const string&, plantType : const string&)
+notify() : void
+~Plant()
Zone
<<use>>
<<use>>
NurseryMediator
-staffMembers : vector<Staff*>
+attach(staff : Staff*) : void
+detach(staff : Staff*) : void
+notify(sender : Staff*) : void
CustomerStaff
+CustomerStaff()
+~CustomerStaff()
+changed() : void
+get() : map<std::string, bool>
+set(message : map<std::string, bool>) : void
+advise(issue : std::string) : void
+customise(plant : BasePlant*, accessory : std::string) : BasePlant*
+getRequestedPlant(plantDetails : Order)
+checkAvailability(name : std::string) : bool
+getFromInventory(plantName : std::string, toUpdate : bool&) : Plant
Manager
<<use>>
<<use>>

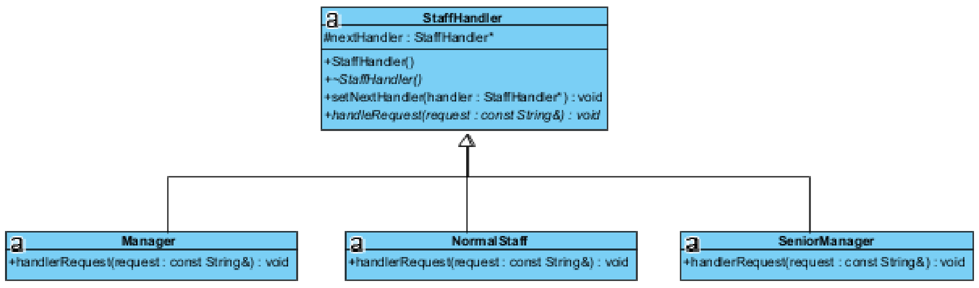
Observer


Staff
-mediator : NurseryMediator
-inv : Inventory*
-stockAvailability : map<std::string, bool>
+Staff()
+~Staff()
+registerMediator(mediator : NurseryMediator*) : void
+deregisterMediator() : void
+changed() : void
+get() : map<std::string, bool>
+set(message : map<std::string, bool>) : void
PlantObserver
#zone : Greenhouse*
+PlantObserver()
+~PlantObserver()
+update() : void
+update(p : Plant*) : void
+setSubject(subject : Greenhouse*) : void
Greenhouse
#observers : vector<PlantObserver*>
+Greenhouse()
+~Greenhouse()
+attach(observer : PlantObserver*) : void
+detach(observer : PlantObserver*) : void
+notify() : void
virtual method such that it may be overwritten in Plant
CareStaff
-insertToInventory(plant : Plant*, toUpdate : bool&) : void
-removeFromInventory(plant : Plant*, toUpdate : bool&) : Plant*
+update(p : Plant*) : void
+changed() : void
+get() : map<std::string, bool>
+set(message : map<std::string, bool>) : void
Plant
+Plant(plant : const Plant&)
+Plant(plantName : const string&, plantType : const string&)
+notify() : void
+~Plant()
Zone
<<use>>
<<use>>

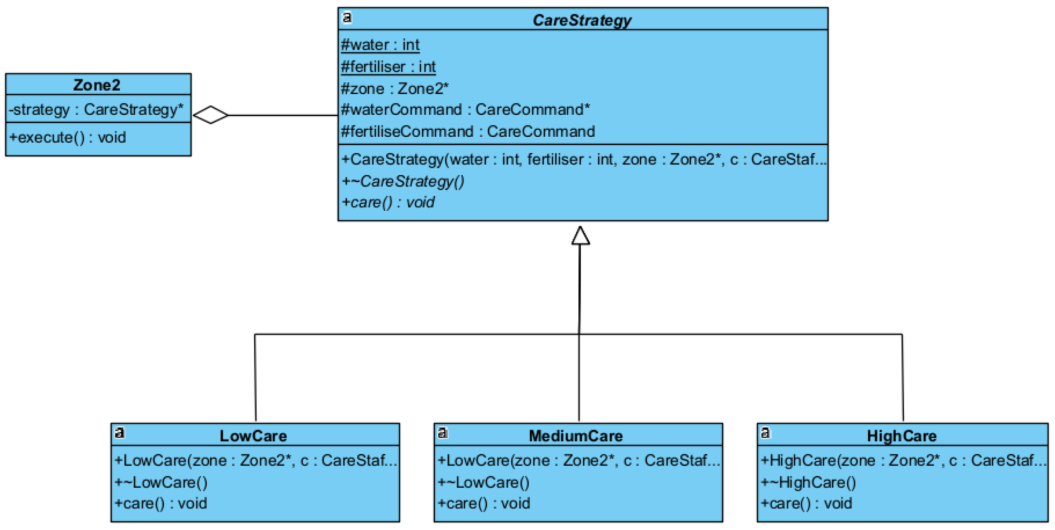
Factory


StaffFactory
+StaffFactory()
+~StaffFactory()
+createStaff() : const Staff*
CareFactory
+createStaff() : const Staff*
CustomerFactory
+createStaff() : const Staff*
Staff
-mediator : NurseryMediator
-inv : Inventory*
-stockAvailability : map<String, bool>
+Staff()
+~Staff()
+registerMediator(mediator : NurseryMediator*) : void
+deregisterMediator() : void
+changed() : void
+get() : map<String, bool>
+set(message : map<String, bool>) : void
CareStaff
+CareStaff()
+~CareStaff()
+performDuty() : void
+water(amount : int) : void
+fertilise(amount : int) : void
-insertToInventory(plant : Plant*, toUpdate : bool&) : void
-removeFromInventory(plant : Plant*, toUpdate : bool&) : Plant*
+update(p : Plant*) : void
+update() : void
+changed() : void
+get() : map<String, bool>
+set(message : map<String, bool>) : void
CustomerStaff
+CustomerStaff()
+~CustomerStaff()
-getFromInventory(plantName : String, toUpdate : bool&) : Plant
+changed() : void
+get() : map<String, bool>
+set(message : map<String, bool>) : void
+advise(issue : String) : void
+customise(plant : BasePlant*, accessory : String) : BasePlant*
+getRequestedPlant(plantDetails : Order)
+checkAvailability(name : String) : bool
<<instantiate>>
<<instantiate>>

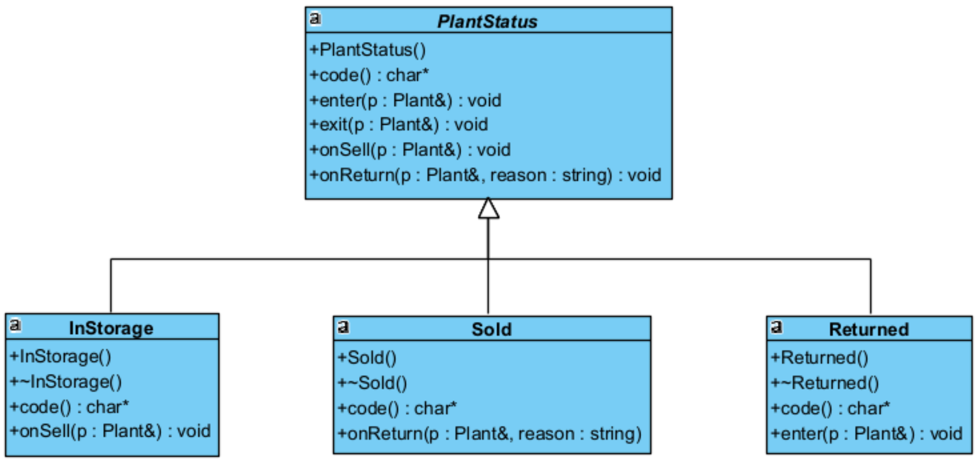
Chain of Responsibility


StaffHandler
#nextHandler : StaffHandler*
+StaffHandler()
+~StaffHandler()
+setNextHandler(handler : StaffHandler*) : void
+handleRequest(request : const String&) : void
Manager
+handlerRequest(request : const String&) : void
NormalStaff
+handlerRequest(request : const String&) : void
SeniorManager
+handlerRequest(request : const String&) : void

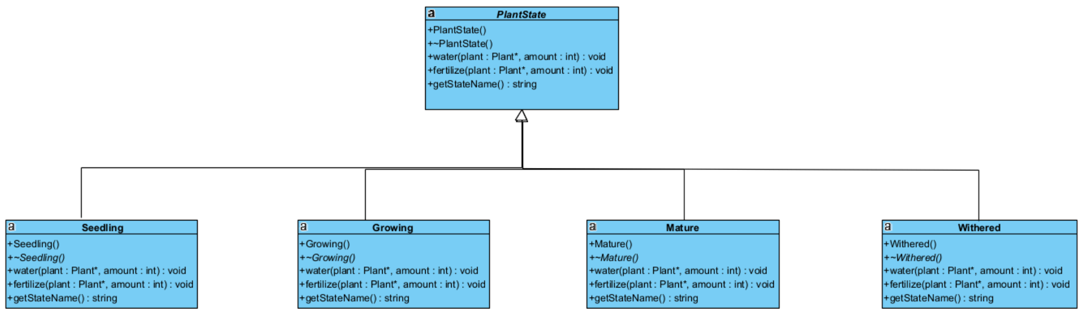
Strategy


Zone2
-strategy : CareStrategy*
+execute() : void
CareStrategy
#water : int
#fertiliser : int
#zone : Zone2*
#waterCommand : CareCommand*
#fertiliseCommand : CareCommand
+CareStrategy(water : int, fertiliser : int, zone : Zone2*, c : CareStaf...
+~CareStrategy()
+care() : void
LowCare
+LowCare(zone : Zone2*, c : CareStaf...
+~LowCare()
+care() : void
MediumCare
+LowCare(zone : Zone2*, c : CareStaf...
+~LowCare()
+care() : void
HighCare
+HighCare(zone : Zone2*, c : CareStaf...
+~HighCare()
+care() : void

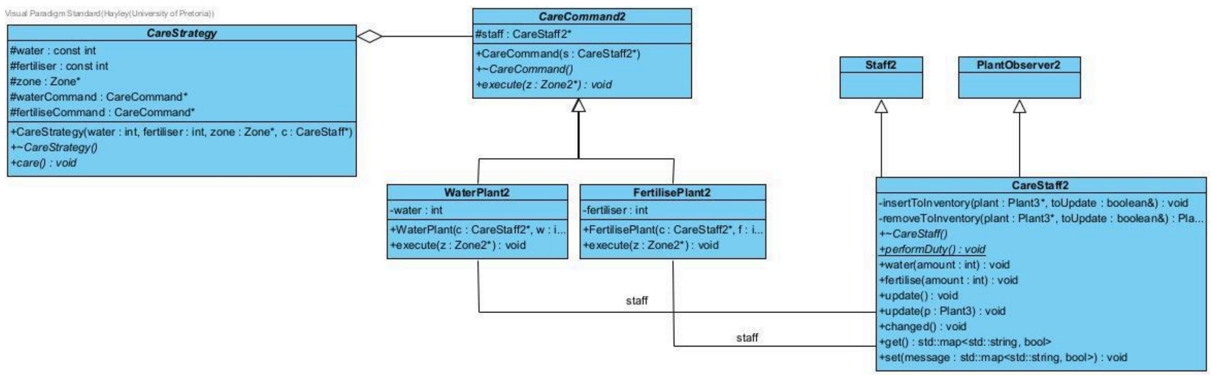
State (Plant Status)



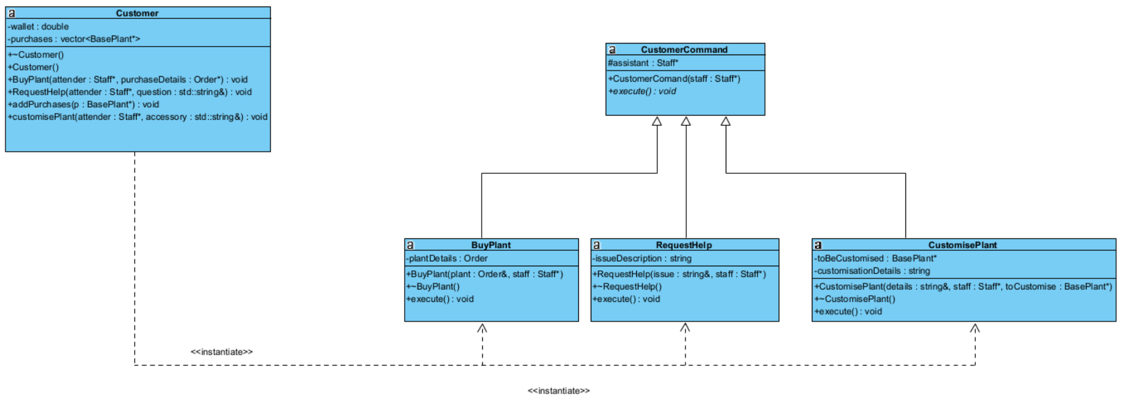
State (Plant State)


PlantState
+PlantState()
+~PlantState()
+water(plant : Plant*, amount : int) : void
+fertilize(plant : Plant*, amount : int) : void
+getStateName() : string
Seedling
+Seedling()
+~Seedling()
+water(plant : Plant*, amount : int) : void
+fertilize(plant : Plant*, amount : int) : void
+getStateName() : string
Growing
+Growing()
+~Growing()
+water(plant : Plant*, amount : int) : void
+fertilize(plant : Plant*, amount : int) : void
+getStateName() : string
Mature
+Mature()
+~Mature()
+water(plant : Plant*, amount : int) : void
+fertilize(plant : Plant*, amount : int) : void
+getStateName() : string
Withered
+Withered()
+~Withered()
+water(plant : Plant*, amount : int) : void
+fertilize(plant : Plant*, amount : int) : void
+getStateName() : string

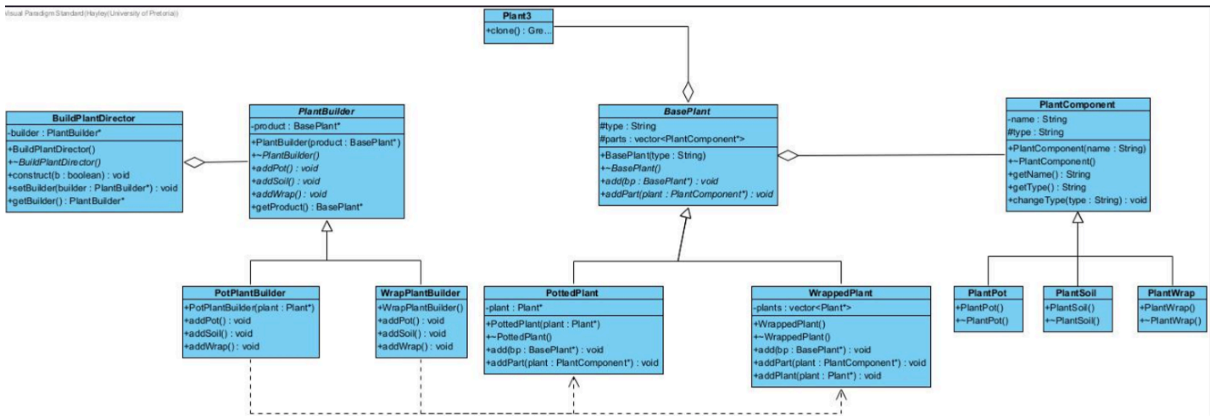
Command (Care Command)


Visual Paradigm Standard(Hayley(University of Pretoria))
CareStrategy
#water : const int
#fertiliser : const int
#zone : Zone*
#waterCommand : CareCommand*
#fertiliseCommand : CareCommand*
+CareStrategy(water : int, fertiliser : int, zone : Zone*, c : CareStaff*)
+~CareStrategy()
+care() : void
CareCommand2
#staff : CareStaff2*
+CareCommand(s : CareStaff2*)
+~CareCommand()
+execute(z : Zone2*) : void
WaterPlant2
-water : int
+WaterPlant(c : CareStaff2*, w : i...
+execute(z : Zone2*) : void
FertilisePlant2
-fertiliser : int
+FertilisePlant(c : CareStaff2*, f : i...
+execute(z : Zone2*) : void
staff
staff
Staff2
PlantObserver2
CareStaff2
-insertToInventory(plant : Plant3*, toUpdate : boolean&) : void
-removeToInventory(plant : Plant3*, toUpdate : boolean&) : Pla...
+~CareStaff()
+performDuty() : void
+water(amount : int) : void
+fertilise(amount : int) : void
+update() : void
+update(p : Plant3) : void
+changed() : void
+get() : std::map<std::string, bool>
+set(message : std::map<std::string, bool>) : void

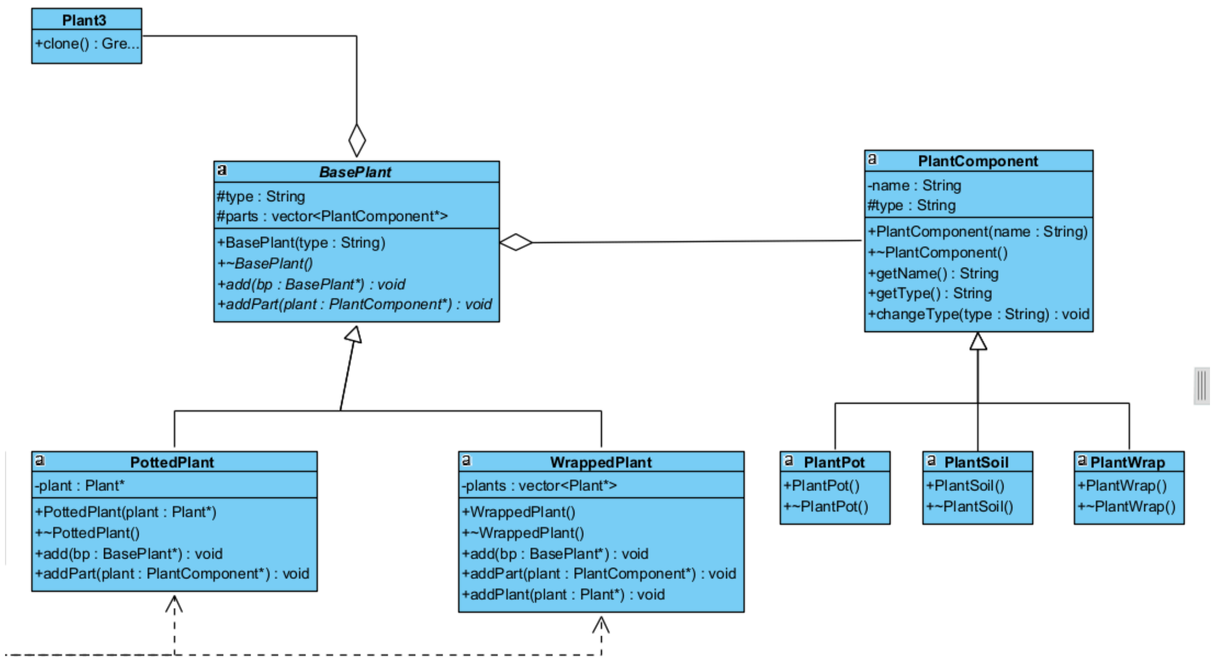
Command (Customer Command)


Customer
-wallet : double
-purchases : vector<BasePlant*>
+~Customer()
+Customer()
+BuyPlant(attender : Staff*, purchaseDetails : Order*) : void
+RequestHelp(attender : Staff*, question : std::string&) : void
+addPurchases(p : BasePlant*) : void
+customisePlant(attender : Staff*, accessory : std::string&) : void
CustomerCommand
#assistant : Staff*
+CustomerComand(staff : Staff*)
+execute() : void
BuyPlant
-plantDetails : Order
+BuyPlant(plant : Order&, staff : Staff*)
+~BuyPlant()
+execute() : void
RequestHelp
-issueDescription : string
+RequestHelp(issue : string&, staff : Staff*)
+~RequestHelp()
+execute() : void
CustomisePlant
-toBeCustomised : BasePlant*
-customisationDetails : string
+CustomisePlant(details : string&, staff : Staff*, toCustomise : BasePlant*)
+~CustomisePlant()
+execute() : void
<<instantiate>>
<<instantiate>>

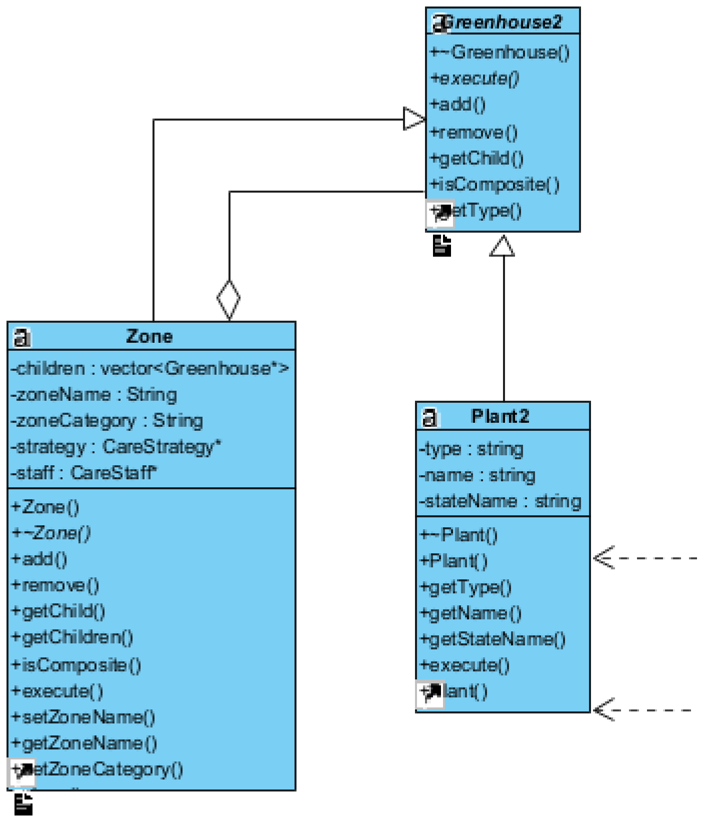
Builder



Decorator

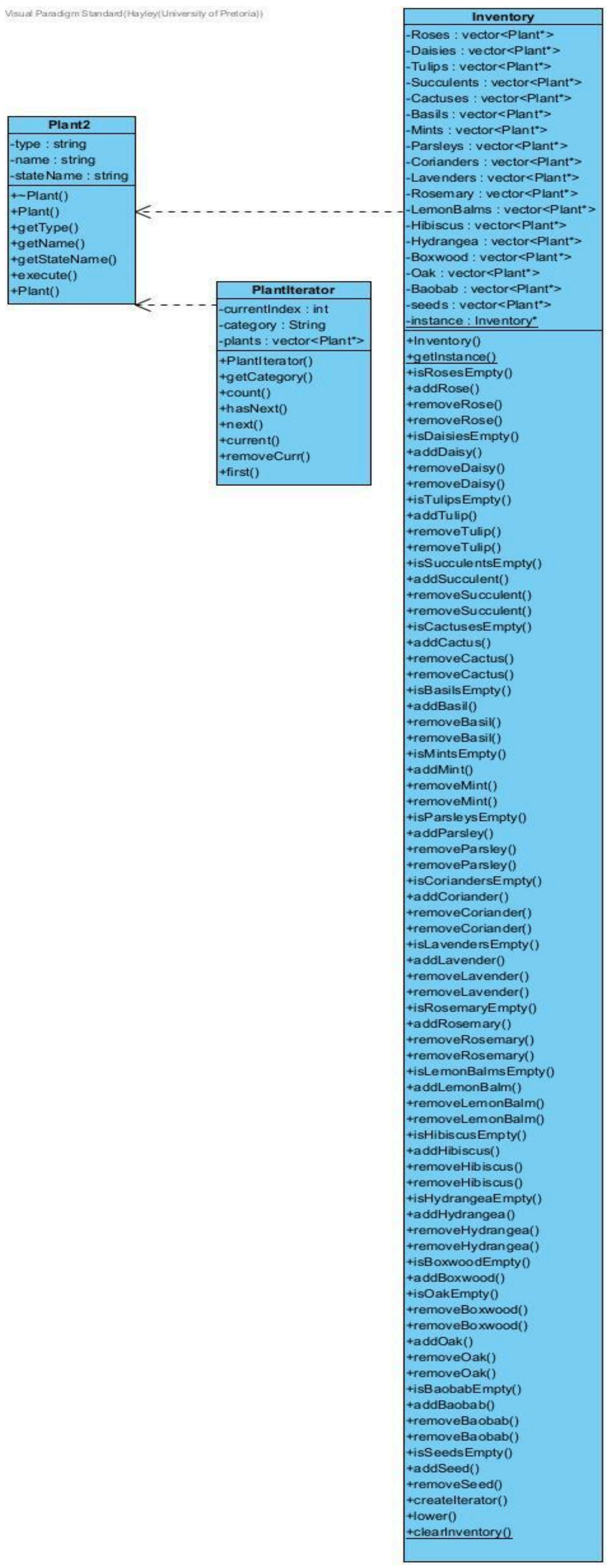

Plant3
+clone() : Gre...
BasePlant
#type : String
#parts : vector<PlantComponent*>
+BasePlant(type : String)
+~BasePlant()
+add(bp : BasePlant*) : void
+addPart(plant : PlantComponent*) : void
PlantComponent
-name : String
#type : String
+PlantComponent(name : String)
+~PlantComponent()
+getName() : String
+getType() : String
+changeType(type : String) : void
PottedPlant
-plant : Plant*
+PottedPlant(plant : Plant*)
+~PottedPlant()
+add(bp : BasePlant*) : void
+addPart(plant : PlantComponent*) : void
WrappedPlant
-plants : vector<Plant*>
+WrappedPlant()
+~WrappedPlant()
+add(bp : BasePlant*) : void
+addPart(plant : PlantComponent*) : void
+addPlant(plant : Plant*) : void
PlantPot
+PlantPot()
+~PlantPot()
PlantSoil
+PlantSoil()
+~PlantSoil()
PlantWrap
+PlantWrap()
+~PlantWrap()

Composite

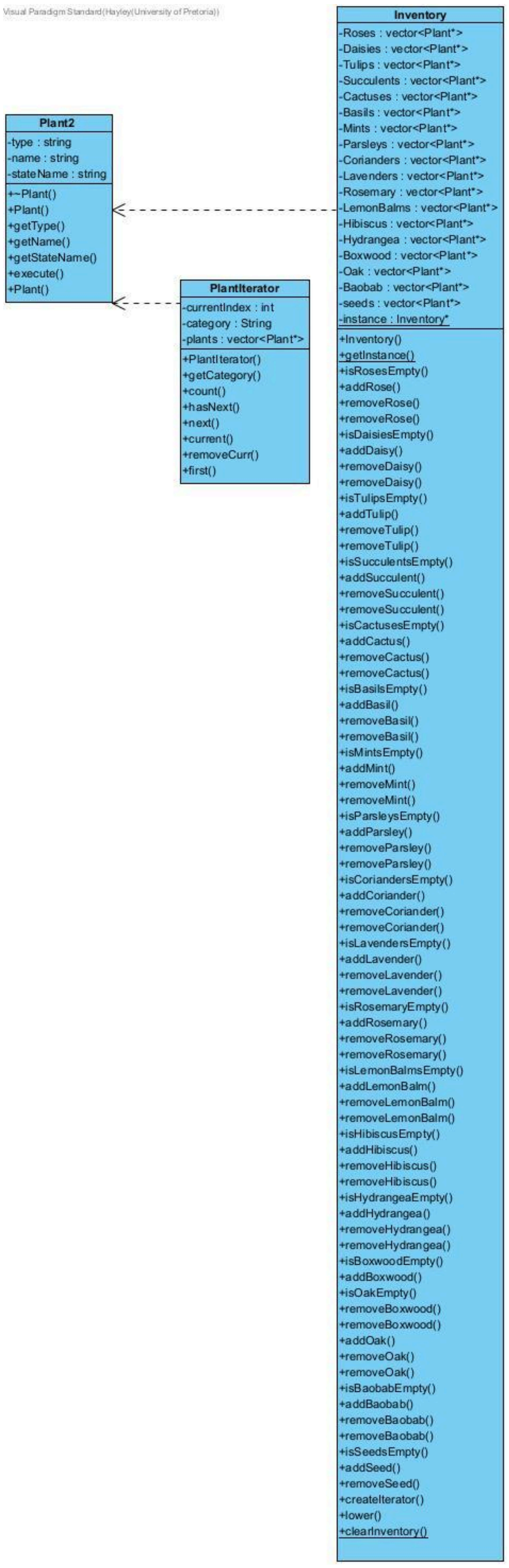


Greenhouse2
+~Greenhouse()
+execute()
+add()
+remove()
+getChild()
+isComposite()
+getType()
Zone
-children : vector<Greenhouse*>
-zoneName : String
-zoneCategory : String
-strategy : CareStrategy*
-staff : CareStaff*
+Zone()
+~Zone()
+add()
+remove()
+getChild()
+getChildren()
+isComposite()
+execute()
+setZoneName()
+getZoneName()
+getZoneCategory()
Plant2
-type : string
-name : string
-stateName : string
+~Plant()
+Plant()
+getType()
+getName()
+getStateName()
+execute()
+Plant()

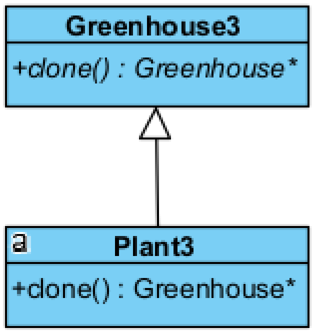
Singleton



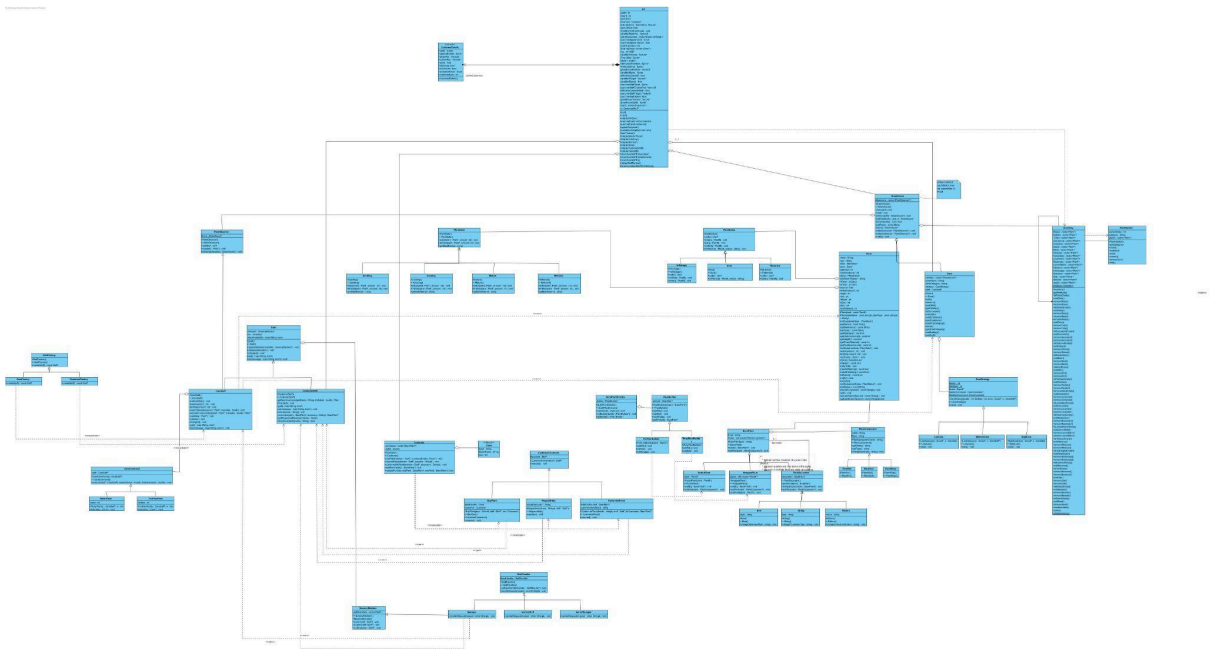
Iterator


Plant2
-type : string
-name : string
-stateName : string
+~Plant()
+Plant()
+getType()
+getName()
+getStateName()
+execute()
+Plant()
PlantIterator
-currentIndex : int
-category : String
-plants : vector<Plant*>
+PlantIterator()
+getCategory()
+count()
+hasNext()
+next()
+current()
+removeCurr()
+first()
Inventory
-Roses : vector<Plant*>
-Daisies : vector<Plant*>
-Tulips : vector<Plant*>
-Succulents : vector<Plant*>
-Cactuses : vector<Plant*>
-Basils : vector<Plant*>
-Mints : vector<Plant*>
-Parsleys : vector<Plant*>
-Corianders : vector<Plant*>
-Lavenders : vector<Plant*>
-Rosemary : vector<Plant*>
-LemonBalms : vector<Plant*>
-Hibiscus : vector<Plant*>
-Hydrangea : vector<Plant*>
-Boxwood : vector<Plant*>
-Oak : vector<Plant*>
-Baobab : vector<Plant*>
-seeds : vector<Plant*>
-instance : Inventory*
+Inventory()
+getInstance()
+isRosesEmpty()
+addRose()
+removeRose()
+removeRose()
+isDaisiesEmpty()
+addDaisy()
+removeDaisy()
+removeDaisy()
+isTulipsEmpty()
+addTulip()
+removeTulip()
+removeTulip()
+isSucculentsEmpty()
+addSucculent()
+removeSucculent()
+removeSucculent()
+isCactusesEmpty()
+addCactus()
+removeCactus()
+removeCactus()
+isBasilsEmpty()
+addBasil()
+removeBasil()
+removeBasil()
+isMintsEmpty()
+addMint()
+removeMint()
+removeMint()
+isParsleysEmpty()
+addParsley()
+removeParsley()
+removeParsley()
+isCoriandersEmpty()
+addCoriander()
+removeCoriander()
+removeCoriander()
+isLavendersEmpty()
+addLavender()
+removeLavender()
+removeLavender()
+isRosemaryEmpty()
+addRosemary()
+removeRosemary()
+removeRosemary()
+isLemonBalmsEmpty()
+addLemonBalm()
+removeLemonBalm()
+removeLemonBalm()
+isHibiscusEmpty()
+addHibiscus()
+removeHibiscus()
+removeHibiscus()
+isHydrangeaEmpty()
+addHydrangea()
+removeHydrangea()
+removeHydrangea()
+isBoxwoodEmpty()
+addBoxwood()
+isOakEmpty()
+removeBoxwood()
+removeBoxwood()
+addOak()
+removeOak()
+removeOak()
+isBaobabEmpty()
+addBaobab()
+removeBaobab()
+removeBaobab()
+isSeedsEmpty()
+addSeed()
+removeSeed()
+createIterator()
+lower()
+clearInventory()

Prototype

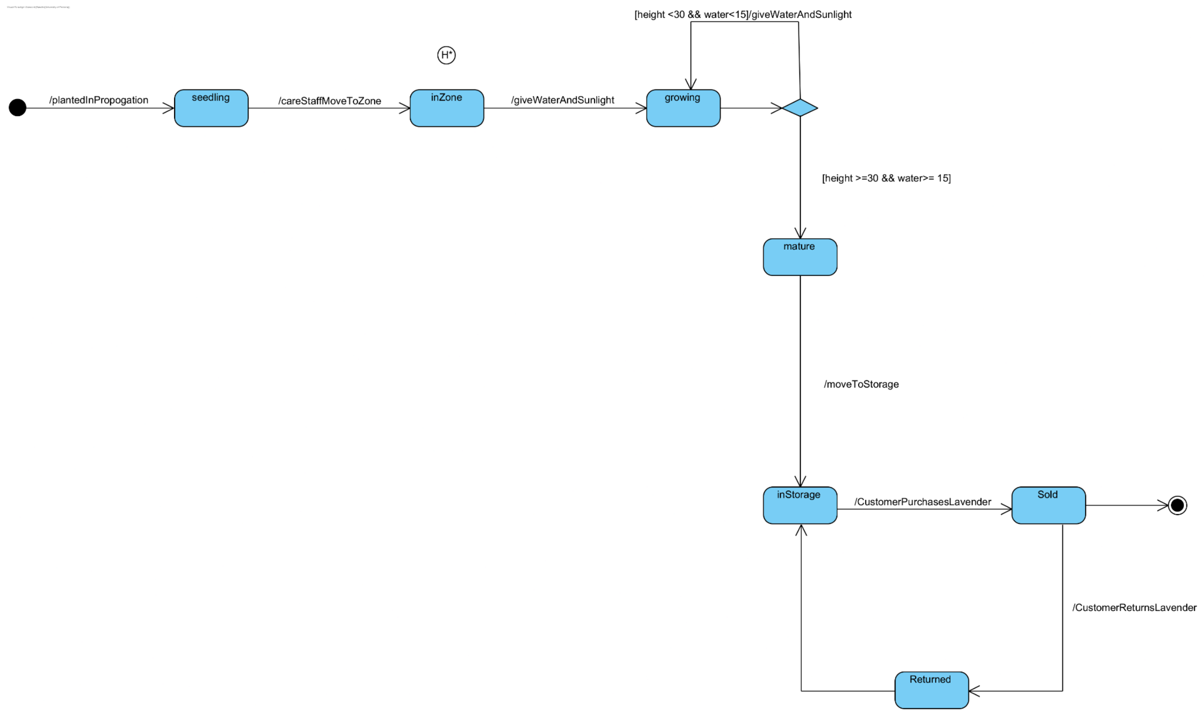


6.2 Class Diagram



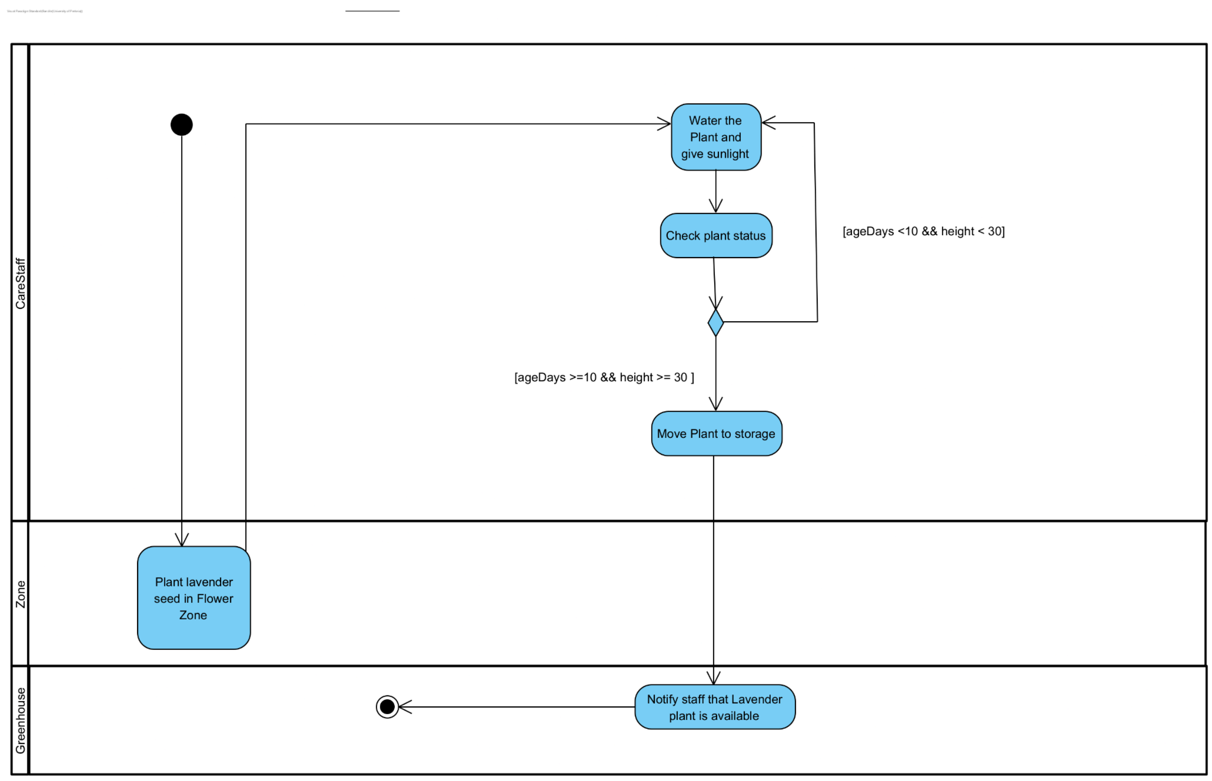
Shows key relationships between Plant, Staff, Customer, Inventory, and Zone classes, and the integration of applied design patterns.

6.3 State Diagram



Illustrates plant transitions between InStorage, Sold, and Returned, including reverse transitions (e.g., when a plant is returned).

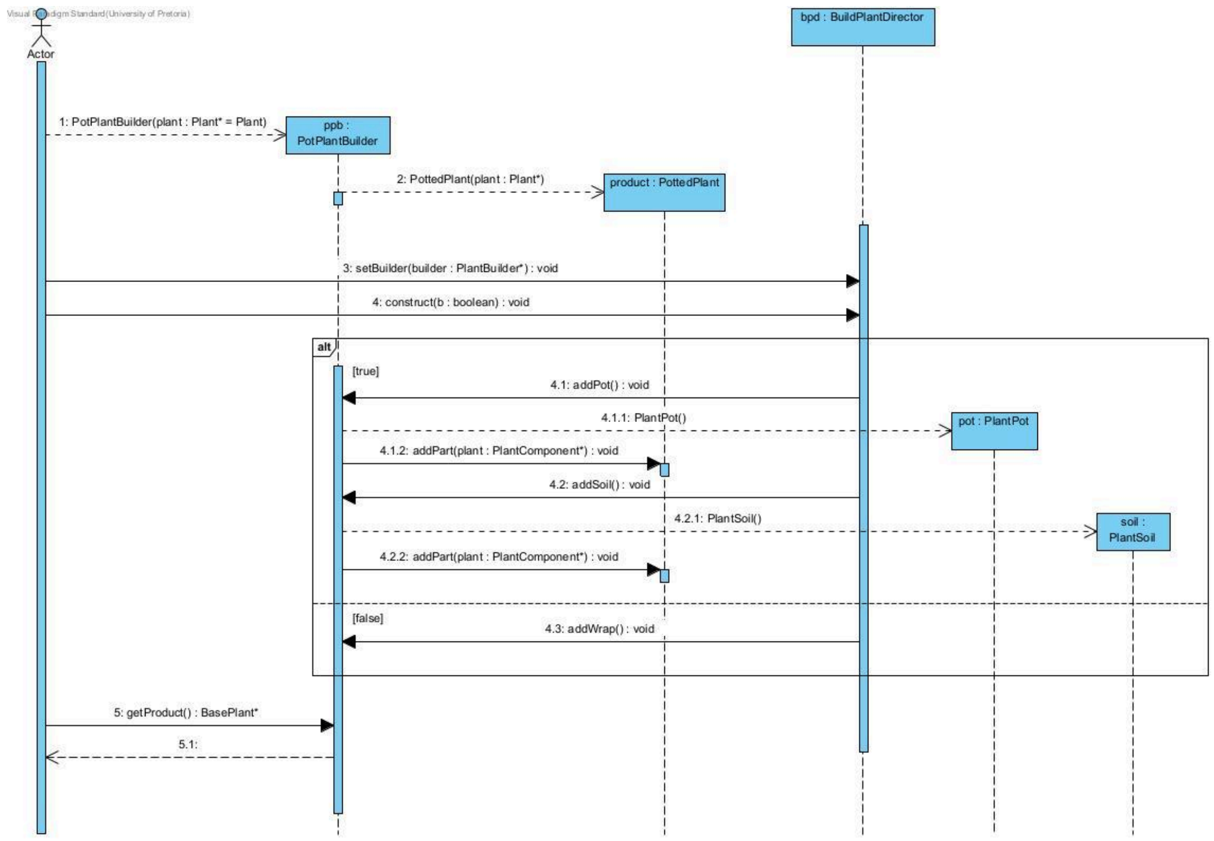
6.4 Activity Diagram



CareStaff
Zone
Greenhouse
Water the Plant and give sunlight
Check plant status
[ageDays <10 && height < 30]
[ageDays >=10 && height >= 30]
Move Plant to storage
Plant lavender seed in Flower Zone
Notify staff that Lavender plant is available

Demonstrates a typical workflow: Create Plant → Apply Care → Notify Staff → Customer Purchase → Update Inventory.

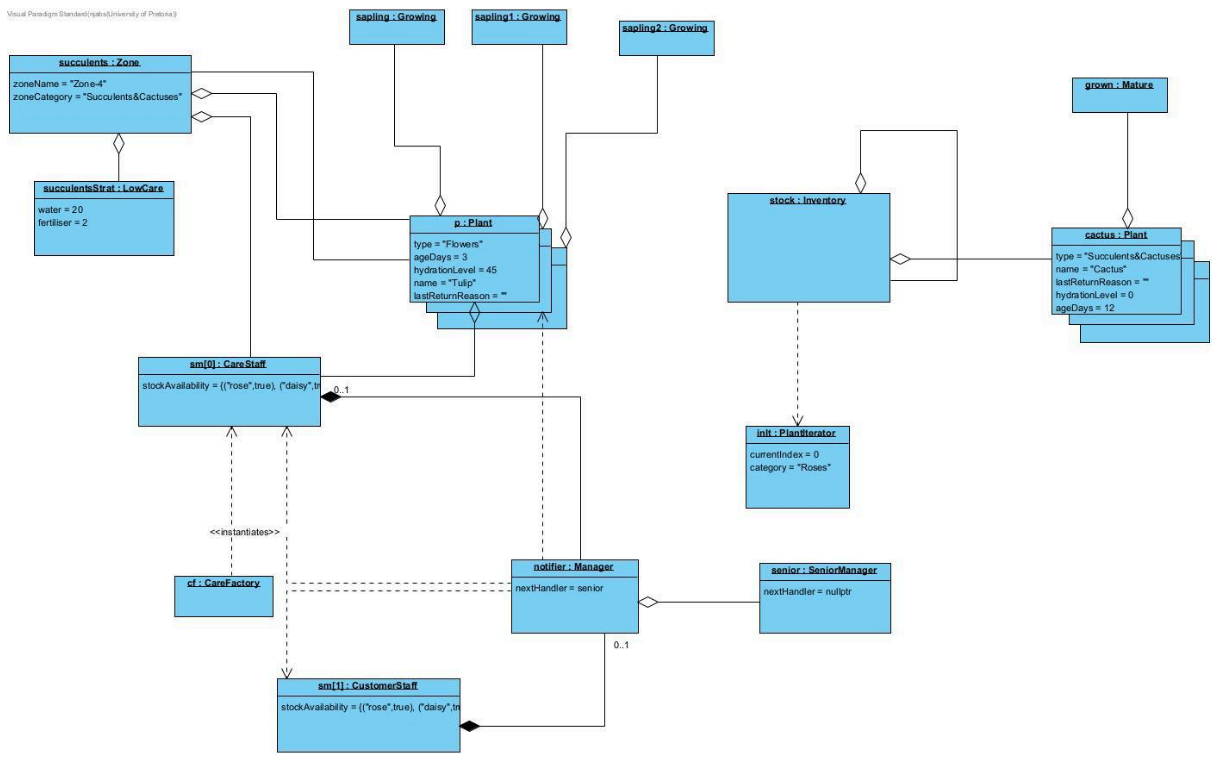
6.5 Sequence Diagram



Visual Paradigm Standard(University of Pretoria)
Actor
ppb : PotPlantBuilder
product : PottedPlant
bpd : BuildPlantDirector
pot : PlantPot
soil : PlantSoil
1: PotPlantBuilder(plant : Plant* = Plant)
2: PottedPlant(plant : Plant*)
3: setBuilder(builder : PlantBuilder*) : void
4: construct(b : boolean) : void
alt
[true]
4.1: addPot() : void
4.1.1: PlantPot()
4.1.2: addPart(plant : PlantComponent*) : void
4.2: addSoil() : void
4.2.1: PlantSoil()
4.2.2: addPart(plant : PlantComponent*) : void
[false]
4.3: addWrap() : void
5: getProduct() : BasePlant*
5.1:

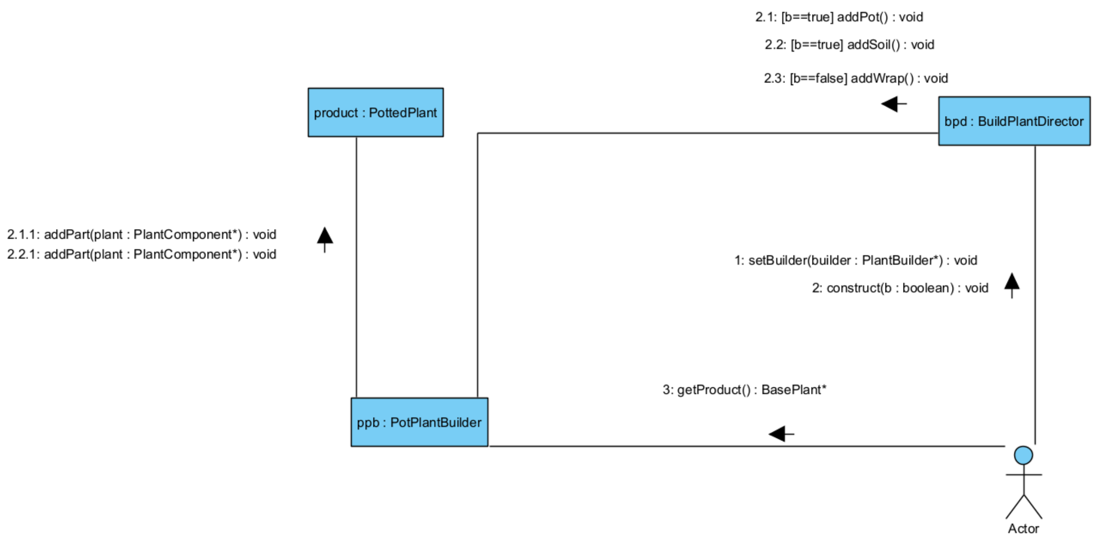
Shows step-by-step message flow between PotPlantBuilder, PottedPlant, BuildPlantDirector, and the PlantPot and PlantSoil classes during the building of a BasePlant product.

6.6 Object Diagram



Represents a snapshot of the system with plants in different states, assigned staff, representing the nursery before customers enter or before nursery opening.

6.7 Communication Diagram


2.1: [b==true] addPot() : void
2.2: [b==true] addSoil() : void
2.3: [b==false] addWrap() : void
product : PottedPlant
bpd : BuildPlantDirector
2.1.1: addPart(plant : PlantComponent*) : void
2.2.1: addPart(plant : PlantComponent*) : void
1: setBuilder(builder : PlantBuilder*) : void
2: construct(b : boolean) : void
3: getProduct() : BasePlant*
ppb : PotPlantBuilder
Actor

Visualise the communication links by step-by-step message flow between PotPlantBuilder, PottedPlant, BuildPlantDirector, and the PlantPot and PlantSoil classes during the building of a BasePlant product.

7. Implementation Summary

The C++23 implementation emphasized modular architecture and strict separation of design pattern logic.
Key classes such as Plant, Zone, Staff, and Inventory form the foundation of the simulation.

7.1 Core Highlights

- Plant Lifecycle: Managed through the State pattern, encapsulating stage-specific behavior.
- Command Execution: Staff execute encapsulated CareCommand objects, enabling flexible scheduling.
- Chain of Responsibility: Ensures requests move up staff hierarchy for appropriate handling.
- Decorator and Builder Integration: Allows user-driven customization of plants.
- Iterator and Composite: Enable easy traversal of grouped plants.
- Observer and Mediator: Maintain synchronized communication between staff, customers, and inventory.

7.2 Testing

The system underwent:

- Unit Tests: Verified behavior of each pattern individually.
- Integration Tests: Checked consistency between modules.
- Edge-Case Tests: Handled null pointers, invalid state transitions, and repetitive commands safely.

---

8. Conclusion and Reflection

The Greenhouse Simulation project demonstrates the effective integration of multiple design patterns into a single coherent system.
By modeling a living digital ecosystem, it showcases how real-world complexity can be represented through modular, reusable software architecture.
Key Lessons Learned:

- Design patterns dramatically improve maintainability and scalability.

- Multiple patterns can coexist harmoniously when responsibilities are clearly separated.

- Collaborative teamwork and iterative testing lead to higher design quality.

Future Enhancements:

- Add a graphical user interface (GUI) for visualization.

- Introduce persistent storage or database integration.

- Expand with AI-based care recommendation features.

---

9. Team Contribution

Name	Student Number	Contribution
Marchant	21549232	Implemented plant lifecycle and state logic as well as the Status of the plant. Created the Communication diagram.
Hayley	24868346	Group Leader and aided with group coordination. Designed Staff hierarchy, Staff Factory, Care Commands and the Chain of Responsibility patterns. Formulated initial and finished the final Class Diagram as well as compiled the report.
Njabulo	24676412	Developed Observer, Mediator, Customer and Customer Commands and Ordering System. As well as helped tremendously with the GUI and code correction. Created the Object Diagram.
Karabo	24865169	Created Builder and Decorator structures for customization, Prototype for easy duplication of seedlings and Care Strategies for Plants in the Nursery. Did most of the system testing. Created the Sequence Diagram.
Bandile	24676394	Created all Iterator, Singleton (Inventory System) and Composite and handled most of the GUI. Created the State and Activity Diagrams.

---

9. References

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
2. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner’s Approach (9th ed.). McGraw-Hill.
3. Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.
4. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.

5. Nanduri, R., & Shetty, S. (2021). Modular Design Patterns in C++. Packt Publishing.
6. South African National Biodiversity Institute (SANBI). (2023). Nursery Management and Plant Propagation. Pretoria, South Africa.