COS 214 Project

git it done

Contents

[Group Member 4](#_Toc212540132)

[A summary of our research 5](#_Toc212540133)

[Functional requirements 6](#_Toc212540134)

[Prototype 7](#_Toc212540135)

[Explanation 7](#_Toc212540136)

[Photo 7](#_Toc212540137)

[Participants 7](#_Toc212540138)

[Iterator 8](#_Toc212540139)

[Explanation 8](#_Toc212540140)

[Photo 8](#_Toc212540141)

[Participants 8](#_Toc212540142)

[Decorator 9](#_Toc212540143)

[Explanation 9](#_Toc212540144)

[Photo 9](#_Toc212540145)

[Participants 9](#_Toc212540146)

[Factory 10](#_Toc212540147)

[Explanation 10](#_Toc212540148)

[Photo 10](#_Toc212540149)

[Participants 10](#_Toc212540150)

[Façade 11](#_Toc212540151)

[Observer 12](#_Toc212540152)

[Explanation 12](#_Toc212540153)

[Photo 12](#_Toc212540154)

[Participants 12](#_Toc212540155)

[Command 13](#_Toc212540156)

[Explanation 13](#_Toc212540157)

[Photo 13](#_Toc212540158)

[Participants 13](#_Toc212540159)

[Strategy 14](#_Toc212540160)

[Explanation 14](#_Toc212540161)

[Photo 14](#_Toc212540162)

[Participants 14](#_Toc212540163)

[State 15](#_Toc212540164)

[Explanation 15](#_Toc212540165)

[Photo 15](#_Toc212540166)

[Participants 15](#_Toc212540167)

[Chain of responsibility 16](#_Toc212540168)

[Explanation 16](#_Toc212540169)

[Photo 16](#_Toc212540170)

[Participants 16](#_Toc212540171)

[Builder 17](#_Toc212540172)

[Explanation 17](#_Toc212540173)

[Photo 17](#_Toc212540174)

[Participants 17](#_Toc212540175)

[Non – functional Requirement 18](#_Toc212540176)

[Security 18](#_Toc212540177)

[Usability 18](#_Toc212540178)

[Scalability 18](#_Toc212540179)

[Performance 18](#_Toc212540180)

[Maintainability 18](#_Toc212540181)

[GUI 18](#_Toc212540182)

# Group Member

u23541475-Joshua Heath

24590739-Antony Van Straten

u23540282-Avuyile Sapula

u24631494-Hamdaan Mirz

u22503685-Dylan Hebron

u24898008-Abdelrahman Ahmed

u24566179-Abdulrahman Sabah

# A summary of our research

# Functional requirements

1. Prototype
2. Iterator
3. Decorator
4. Factory
5. Observer
6. Facade
7. Chain of responsibility
8. Command
9. Strategy
10. State
11. Prototype

## Prototype

### Explanation

This will be utilized when we want to replicate an existing plant exactly. Instead of merely constructing an existing one, which takes time for individuals in general, this will help the system simplify things.

### Photo

A blue rectangular object with black text

AI-generated content may be incorrect.

### Participants

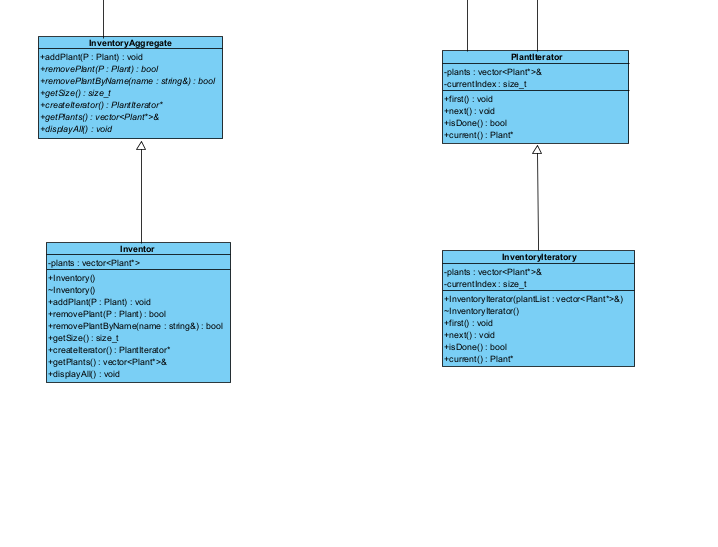
|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Prototype | ? |
| ConcretePrototype | ? |
| Client | ? |
|  |  |
|  |  |

## Iterator

### Explanation

* In order for the plant iterator to traverse through the entire inventory and verify every plant that was added by the staff, I will only be able to add a single plant to the iterator.
* Therefore, in order to properly iterate through the Inventory class, we will include functions like first(), next(), isDone(), and current() in our methods.

### Photo



### Participants

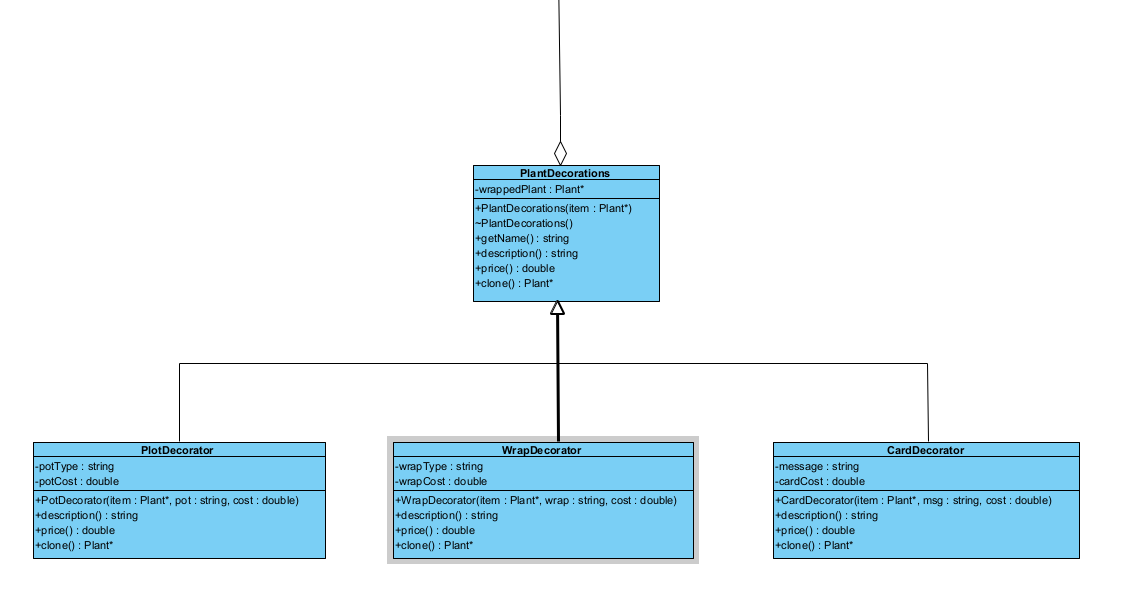
|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Iterator | PlantIterator |
| ConcreteIterator | InventoryIterator |
| Aggregate | ? |
| ConcreteAggregate | ? |

## Decorator

### Explanation

Thus, when a consumer wants to purchase something, we will provide them with additional features like a plot decorator, a wrap decorator, and a card decorator. They will be able to select the kind of decoration they like to add when they purchase the plant.

### Photo



### Participants

|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Component | Plant |
| ConcreteComponent | Cactus |
| ConcreteComponent | Lavender |
| ConcreteComponent | Rose |
| Decorator | PlantDecorations |
| ConcreteDecoratorA | ? |
| ConcreteDecoratorB | ? |

## Factory

### Explanation

* The primary duty of the plant factory is to produce various plant varieties for our project.
* So it follows a simple factory Design pattern, where a single factory class is used to produce multiple plant objects depending on the requested type.

### Photo

A blue rectangular object with black lines

AI-generated content may be incorrect.

### Participants

|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Creator | Creator |
| ConcreteCreator | CactusCreator |
| ConcreteCreator | LavenderCreator |
| ConcreteCreator | RoseCreator |
| Product | Plant |
| ConcreteProduct | Cactus |
| ConcreteProduct | Lavender |
| ConcreteProduct | Rose |

## Façade

STILL NEEDS TO BE DONE

## Observer

### Explanation

### Photo

### Participants

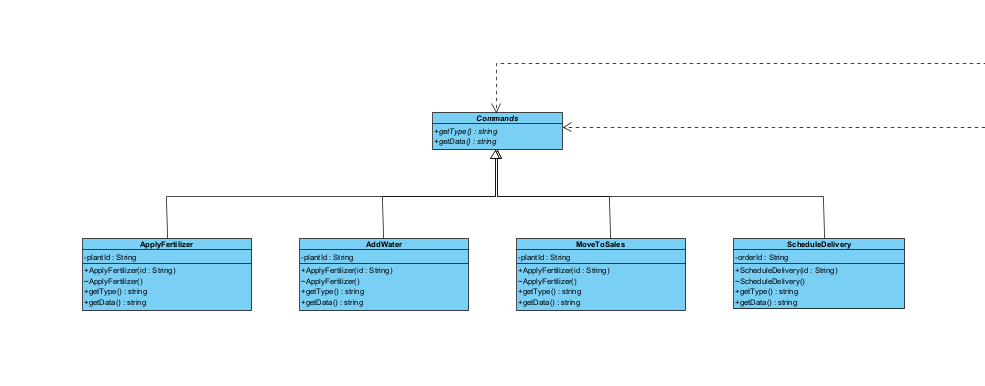
|  |  |
| --- | --- |
| **Participant** | **Class name** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Command

Our commands, which are distinct objects, will encapsulate actions or requests.   
The following courses, ApplyFertilizer, will be utilized when a plant requires fertilizer. When water is needed for the plants, we also have Add Water. Additionally, there will be a command called MoveToSale that may be used to move a plant to sales so that customers can buy it. Additionally, we will have ScheduleDelivery so that the employees are aware that the customer needs this plant supplied.

### Explanation

### Photo



### Participants

|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Command | Commands |
| ConcreteCommand | ApplyFertilizer |
| ConcreteCommand | AddWater |
| ConcreteCommand | MoveToSales |
| ConcreteCommand | ScheduleDelivery |
| Client | ? |
| Invoker | ? |
| Receiver | ? |

## Strategy

### Explanation

This class shows how different plants have different ways to be cared for

The project uses the Strategy Pattern, where each type of plant has its own "care plan" or strategy, rather than writing all the care logic inside a single large class.   
Thus, we'll have the following We will have RoseCare, a specific care type for roses, CactusCare, a specific care type for cacti, and SucculentCare, a specific care type for succulents.

### Photo

A screenshot of a computer

AI-generated content may be incorrect.

### Participants

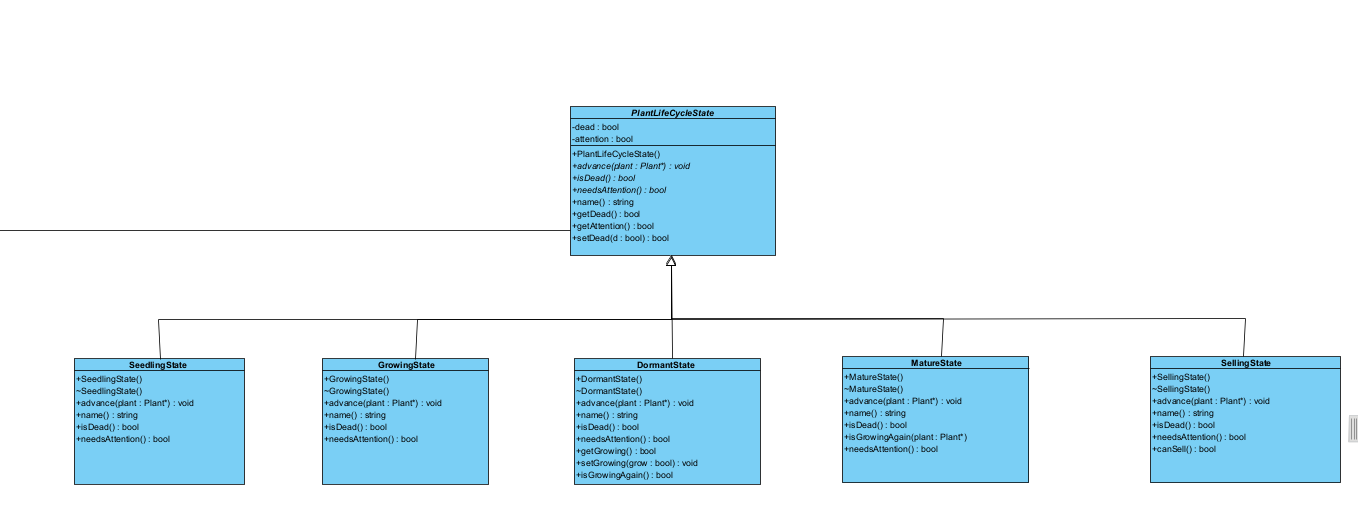
|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Strategy | CareStrategy |
| ConcreteStrategy | CactusCare |
| ConcreteStrategy | LavenderCare |
| ConcreteStrategy | RoseCare |
| Context | Plant |

## State

Therefore, when an object's behaviour modifies its present state, the state pattern will be applied.   
For our project, the plant will go through multiple phases, like a life cycle:

Seedling → Growing → Dormant → Mature → Selling

### Explanation



### Photo

### Participants

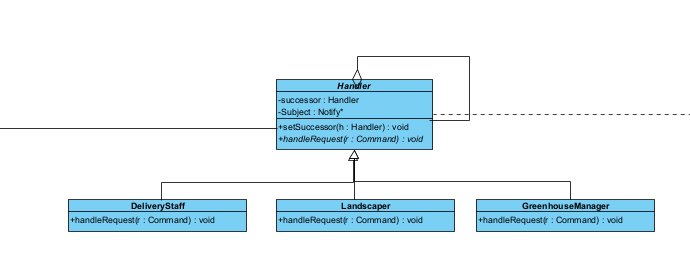
|  |  |
| --- | --- |
| **Participant** | **Class name** |
| State | PlantLifeCycleState |
| ConcreteState | SeedlingState |
| ConcreteState | GrowingState |
| ConcreteState | DormantState |
| ConcreteState | MatureState |
| ConcreteState | SellingState |
| Context | Plant |

## Chain of responsibility

### Explanation

As a result, the requests, which are commands, will be able to move up the chain of command until one of them—DeliveryStaff, LandScaper, or GreenHouseManager—can handle them. All handlers will use the handler class as their base class.

### Photo



### Participants

|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Handler | Handler |
| Concrete Handler | DeliveryStaff |
| Concrete Handler | Landscaper |
| Concrete Handler | GreenhouseManager |
| Client | ? |

## Builder

### Explanation

So the builder will be used to construct complex plant object step by step- for example, building a Rose or a Succulent with specific care strategies and life cycle states.

### Photo

A screenshot of a computer flowchart

AI-generated content may be incorrect.

### Participants

|  |  |
| --- | --- |
| **Participant** | **Class name** |
| Builder | PlantBuilder |
| Concrete Builder | CactusBuilder |
| Concrete Builder | LavenderBuilder |
| Concrete Builder | RoseBuilder |
| Director | GreenhouseDirector |
| Product | Plant |
| Product | Cactus |
| Product | Lavender |
| Product | Rose |

# Non – functional Requirement

## Security

* Only employees are permitted to add plants; clients are not permitted to do so.
* Only the items(Plants) will be accessible from the greenhouse.
* To prevent customers from entering, we will provide the staff with a short password.
* Data about customers, staff, and plant inventory should be securely stored and protected from unauthorized access.

## Usability

* Both customers and employees will find the user interface to be tidy and pleasant.
* Both customers and employees will find the user interface to be tidy and pleasant.
* When a system error occurs, the consumer should receive clear instructions and feedback.

## Scalability

* There will be a limit on how many plants the greenhouse can hold.

## Performance

* If the consumer makes a mistake, like inputting the incorrect password, the response will be returned to him.
* Additionally, the client will be added to view the update after the data is added from the stuff, and the user logs out.

## Maintainability

* For them to know whether to add or remove a function in the event of a merge dispute, we will be documenting our code.
* In our project, unit testing will be used. Every individual will conduct their own unit testing.
* We will also be adding Github Actions Linter and Tester for bonus marks.

## GUI

* We will utilize our C++ code as our API and use HTML and JavaScript to construct a graphical user interface.
* To connect and view our pages online, we will also need a server.cpp file.