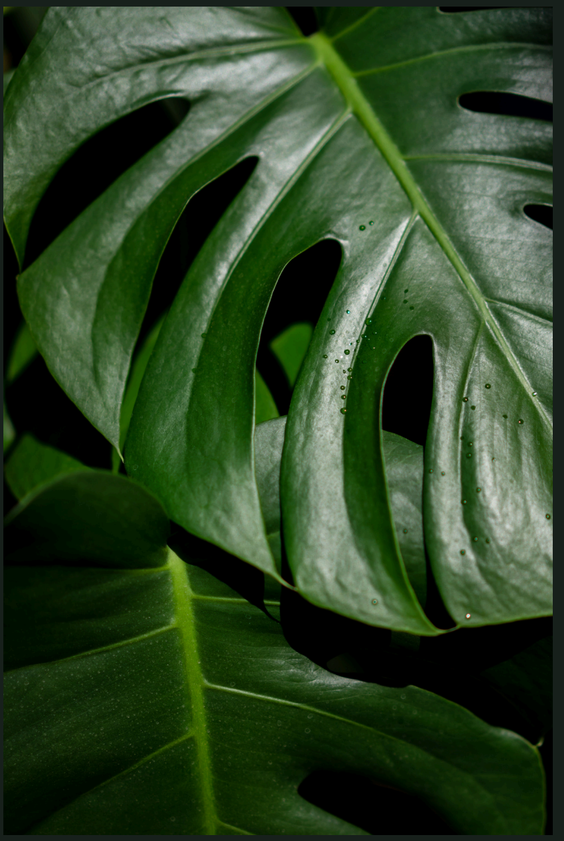
GREENHOME
-BY STARTING 5

COS 214
GROUP
PROJECT



MEET THE 5

Mosa Leiee - 24735672
Lesedi Padi - u24096017
Brayden Butler - u24824713
Kundai Ndemera - u23941996
Sentelweyinkhosi Mngomezulu - u24874478

ASSUMPTIONS

- We have multiple Greenhouses that utilize the same Inventory.
- The workflow demoed in main does not support concurrent customer and admin functionalities.
- The care strategies are suitable estimates for each specific type of plant (e.g. Flower, Tree, Succulent).

FUNCTIONAL REQUIREMENTS

Greenhouse - Plant Management

Strategy pattern - used to apply different care strategies to plants based on their type, making plant care more flexible and adaptable.

State pattern - allows the system to have plants in different states to demonstrate plant lifecycle management.

Factory Method - uses different factories to create different types of plants essential for the nursery.

Prototype pattern - allows the system to make multiple plants or bundles based off of others.

Staff Operations

Command pattern - allows plant care and sales to be encapsulated as objects and to be reused as commands.

Observer pattern - Notifies employees of changes in plant state or inventory counts.

Mediator pattern - allows customers to communicate with staff for information, recommendations and orders

Customer and Sales Floor

Composite pattern - allows for plants to be grouped together for decorative and sales purposes.

Decorator pattern - allows for plants/plant groups to be given modifiers such as plant wrappings and decorative pots.

Singleton - ensures there is only one inventory for multiple greenhouses at any given moment to avoid inconsistencies.

OUR PROGRAM

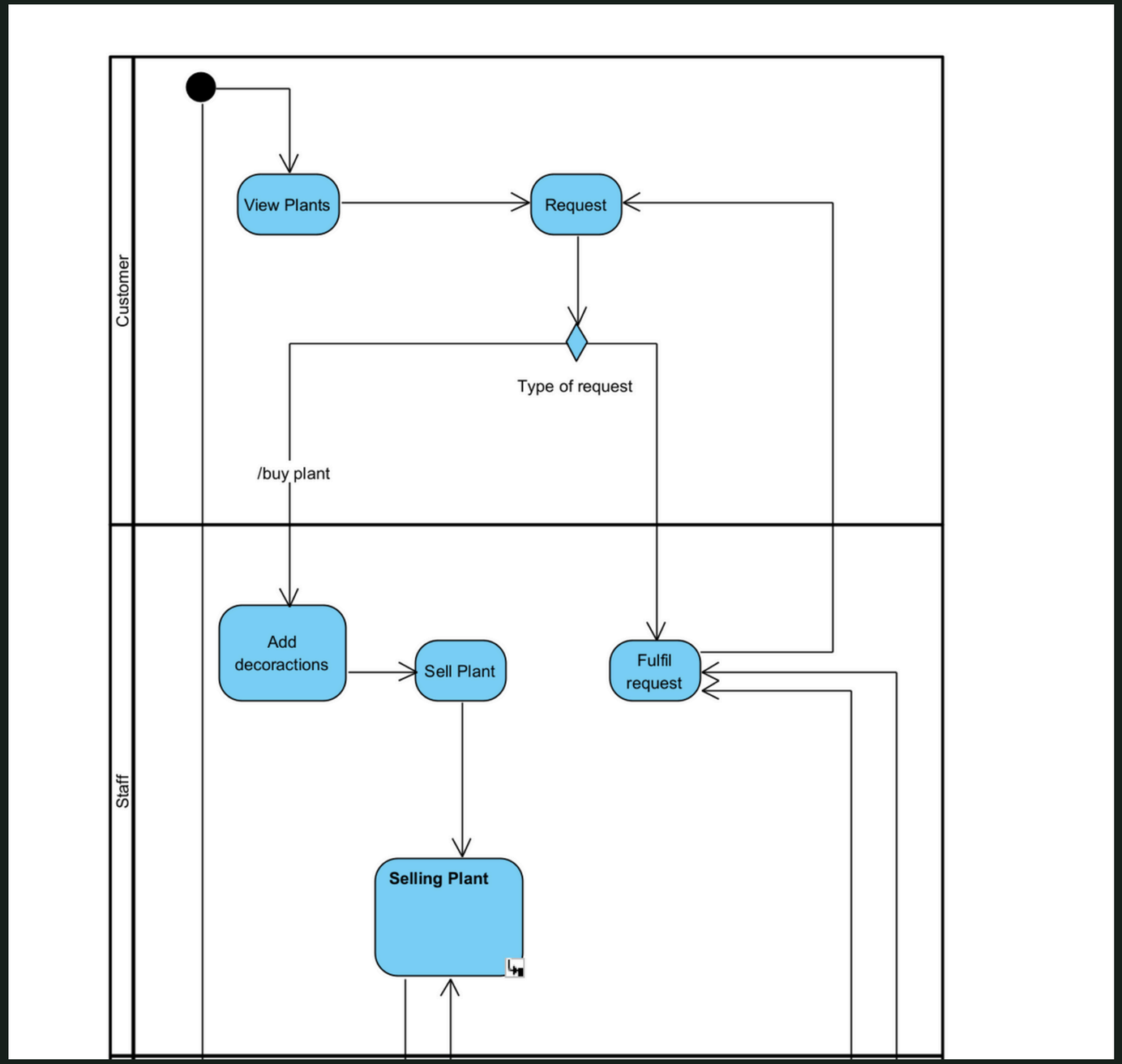
Our interactive Plant Nursery Simulator empowers users to manage a dynamic nursery ecosystem through hands-on decision-making, with automated systems supporting plant growth and customer interactions.

The Core:

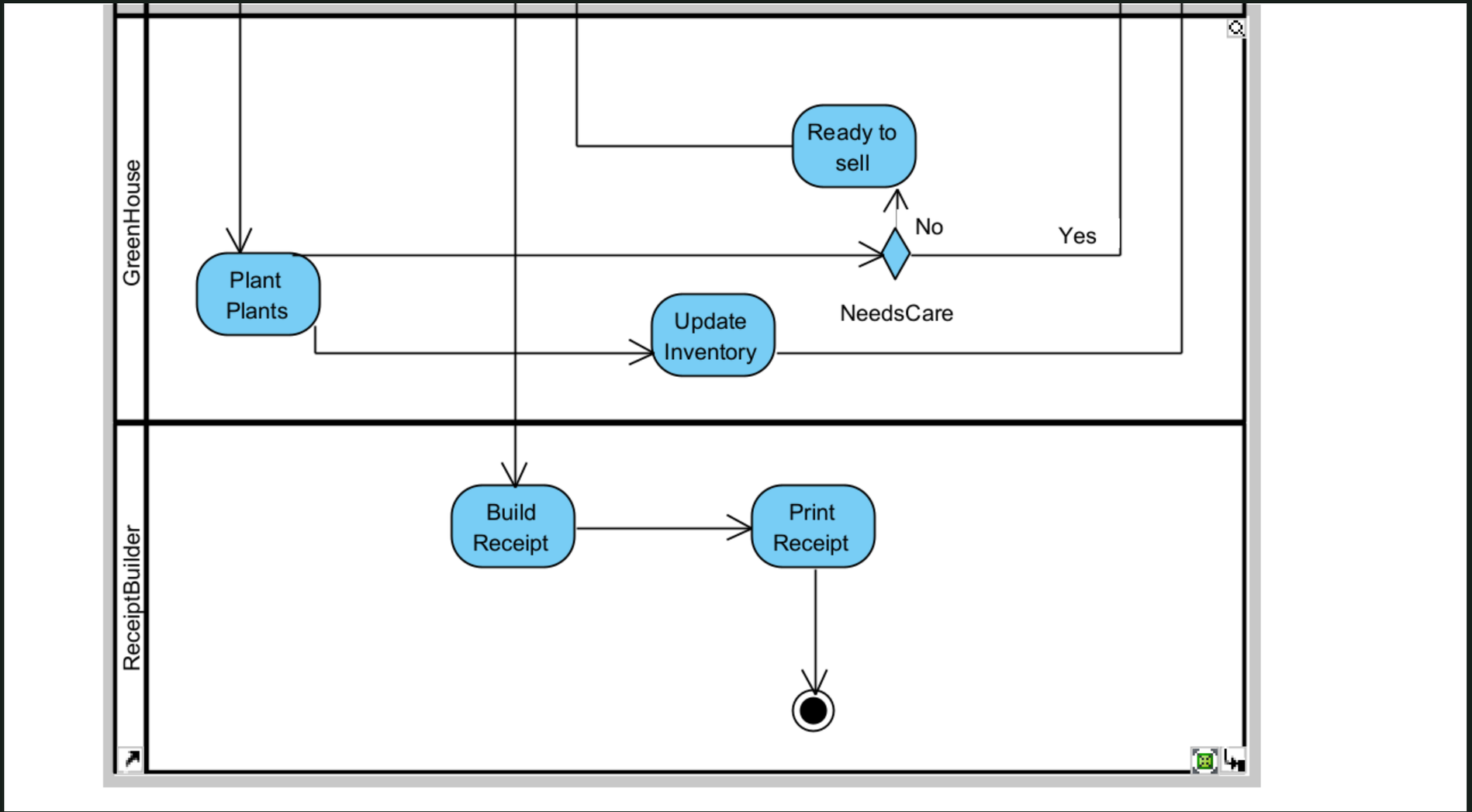
- Plant Health and Lifecycle States (Seed → Sprout → Mature → Death)
- Inventory Levels and Stock Changes
- Staff Request Processing and Task Completion
- Customer Interaction History
- Resource Allocation (Water, Sunlight via Commands)

The system blends user control through a text-based menu with pattern-driven automation. Users initiate actions but the underlying design patterns handle routing, notifications, state management and consistent behaviour without manual intervention.

ACTIVITY DIAGRAM

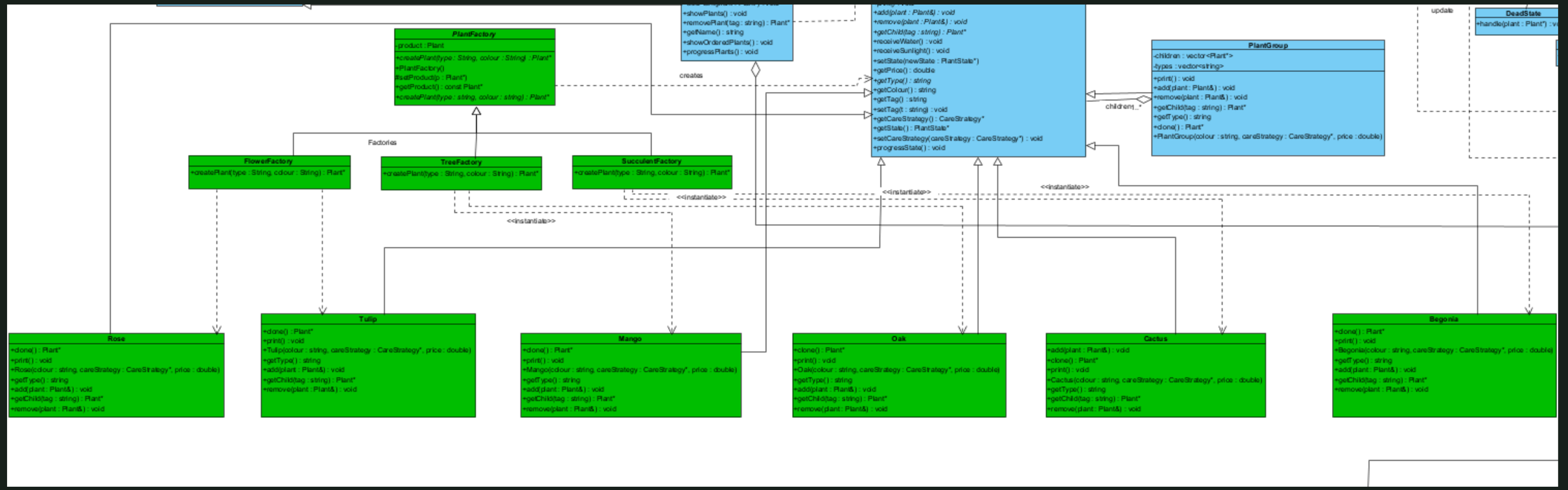


ACTIVITY DIAGRAM



CLASS DIAGRAM

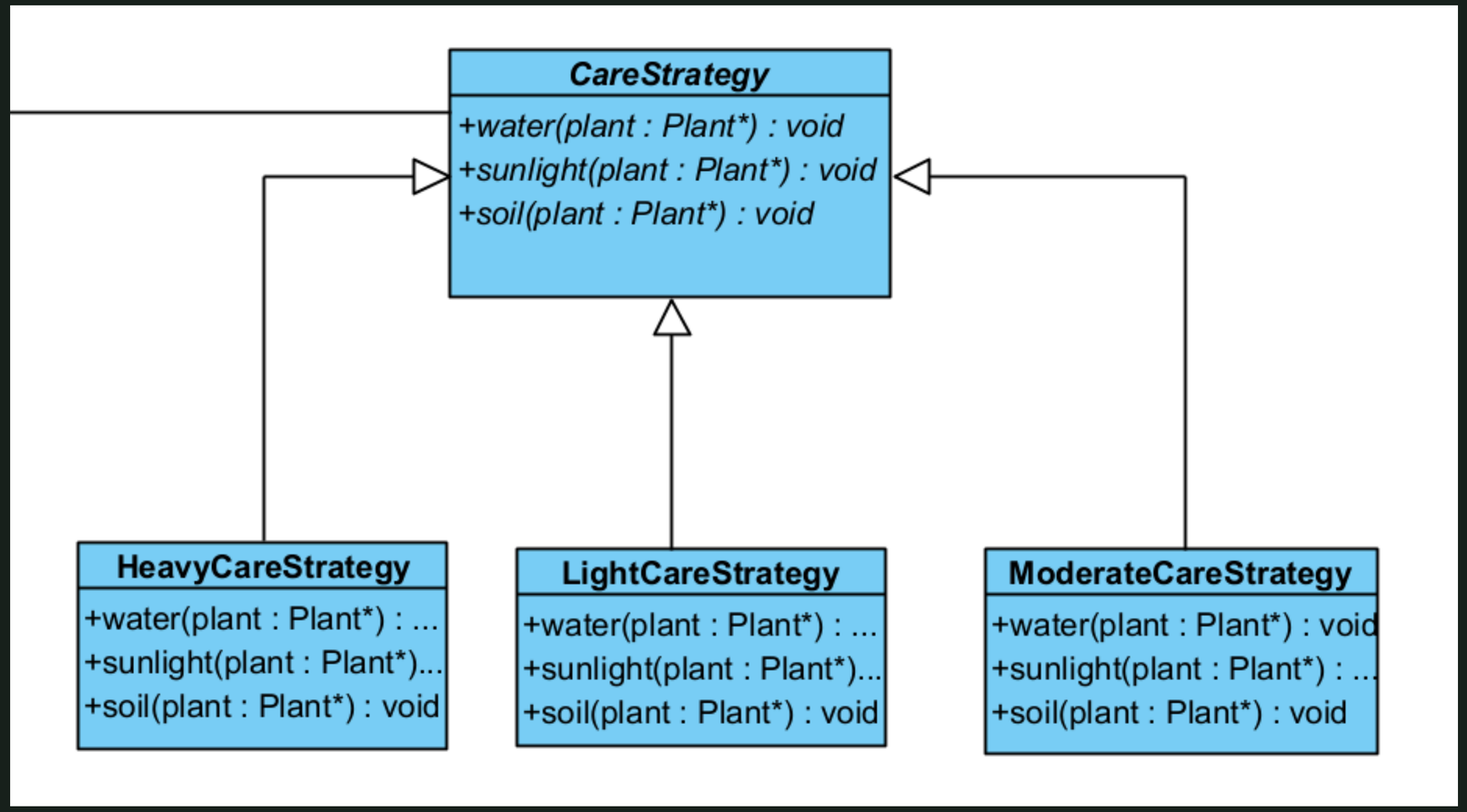
FACTORY METHOD CLASS DIAGRAM



PARTICIPANTS:

Creator: PlantFactory
ConcreteCreator: FlowerFactory, TreeFactory and SucculentFactory
Product: Plant
ConcreteProduct: Rose, Tulip, Oak, Mango, Begonia and Cactus

STRATEGY CLASS DIAGRAM



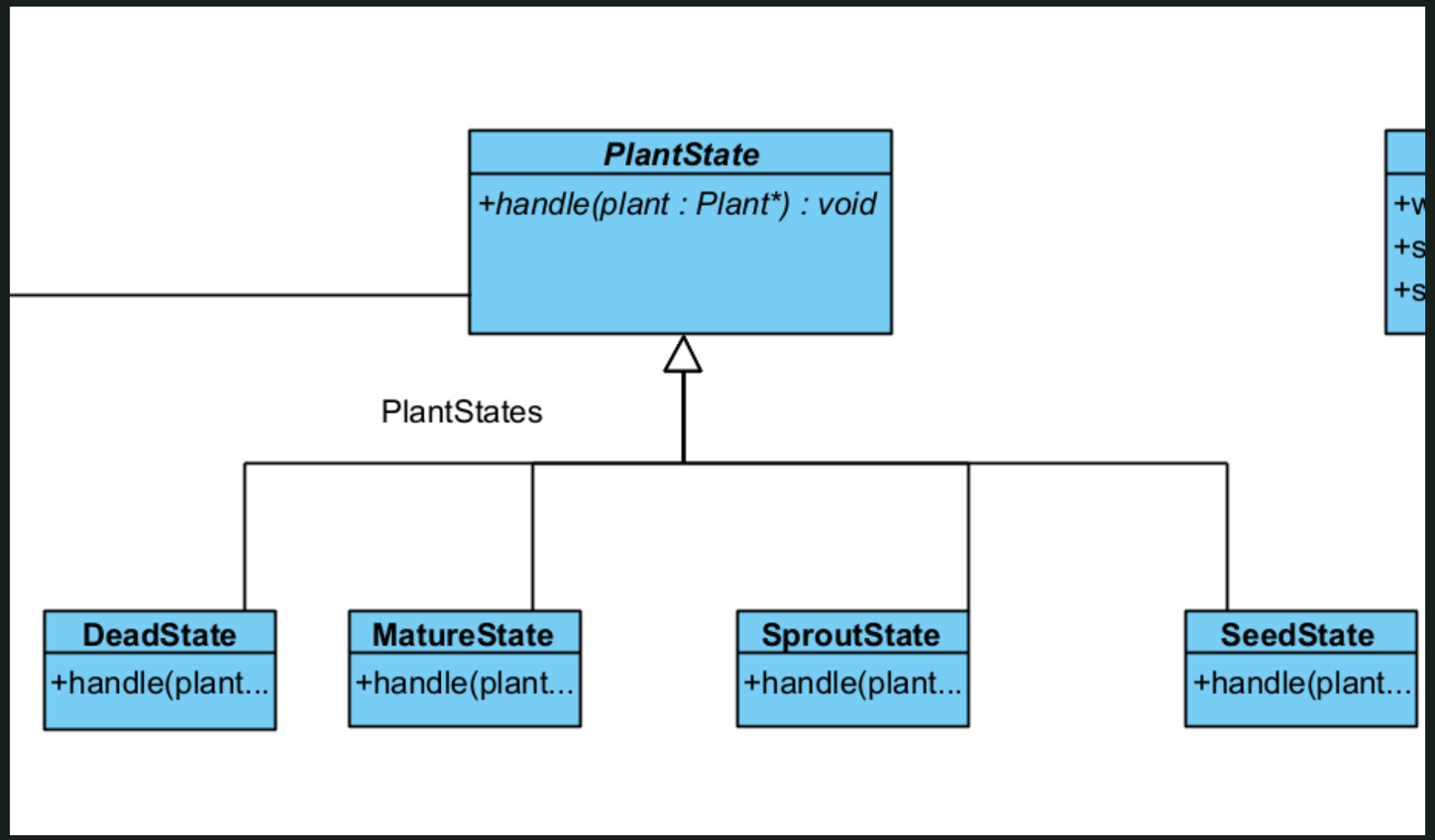
Participants:

Strategy: CareStrategy

ConcreteStrategy: HeavyCareStrategy, LightCareStategy and ModerateCareStrategy

Context: Plant

STATE CLASS DIAGRAM



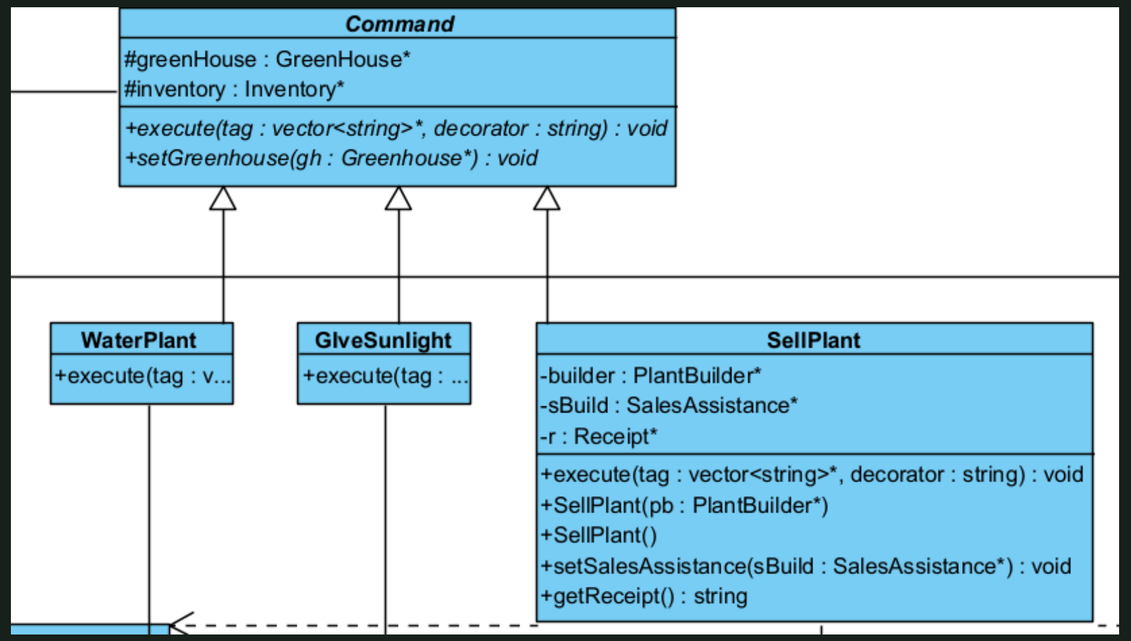
Participants:

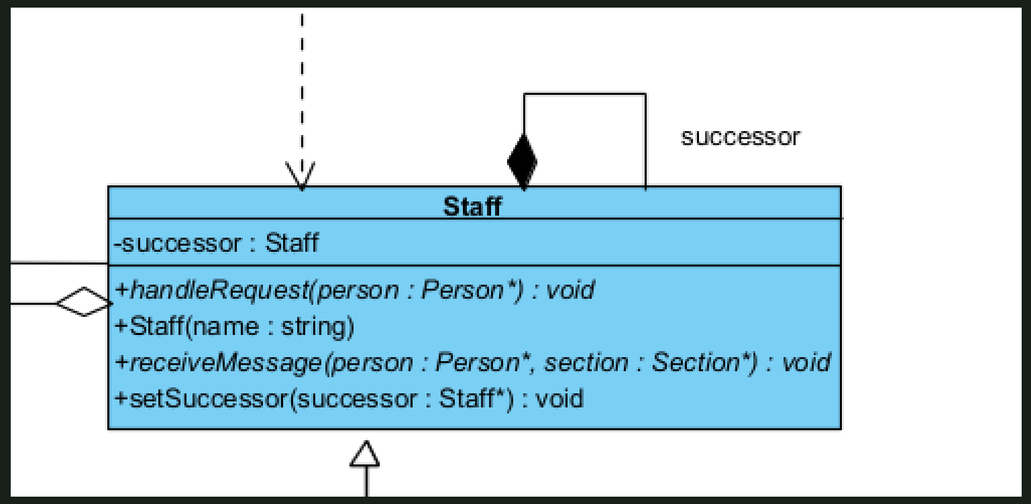
State: CareStrategy

ConcreteState: SeedState, SproutState, MatureState and DeadState

Context: Plant

COMMAND CLASS DIAGRAM





PARTICIPANTS:

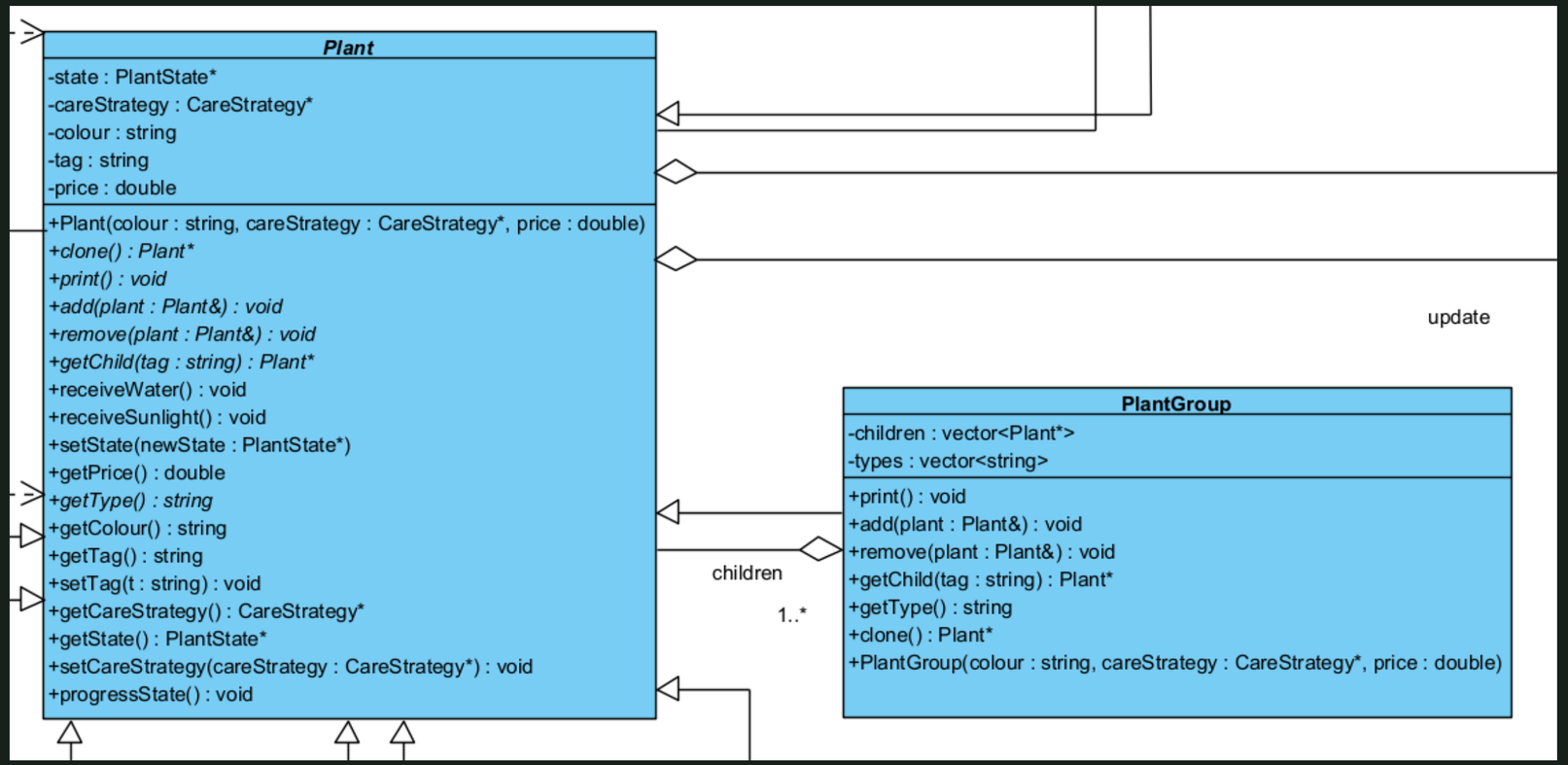
Invoker: Staff

Command: Command

ConcreteCommand: WaterPlant, GiveSunlight and SellPlant

Reciever: Greenhouse

COMPOSITE AND PROTOTYPE CLASS DIAGRAM



PARTICIPANTS:

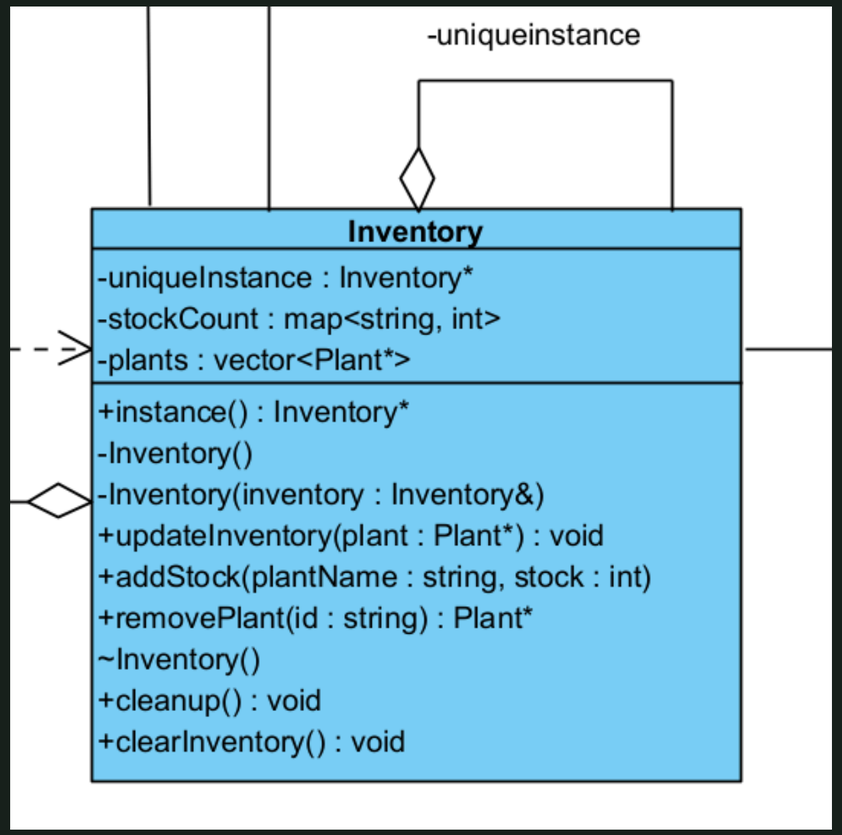
Component: Plant

Leaf:
Rose,Tulip,Begonia,Mango,Oak,Cactus

Composite: PlantGroup

Prototype is also here via the clone() function
(Prototype: Plant; ConcretePrototype: rose,tullip,oak etc.)

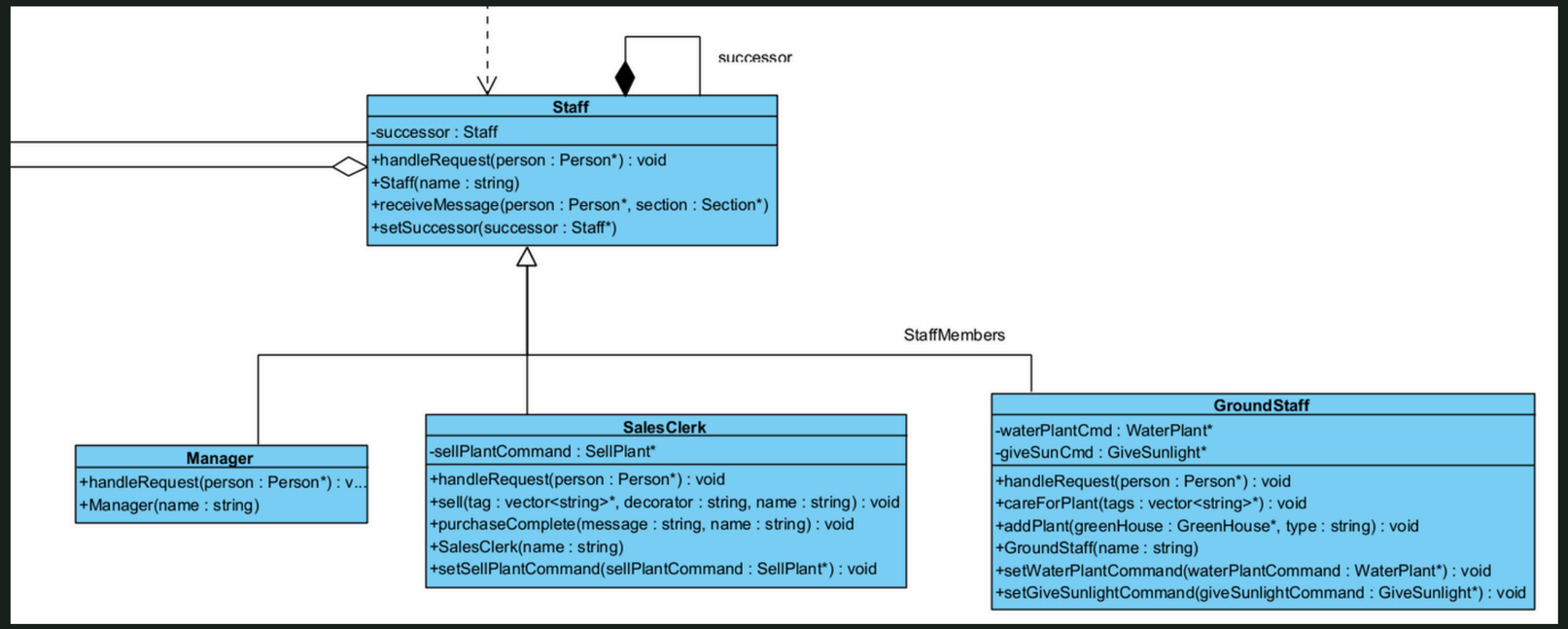
SINGLETON CLASS DIAGRAM



PARTICIPANTS:

Singleton: Inventory

CHAIN OF RESPONSIBILITY CLASS DIAGRAM

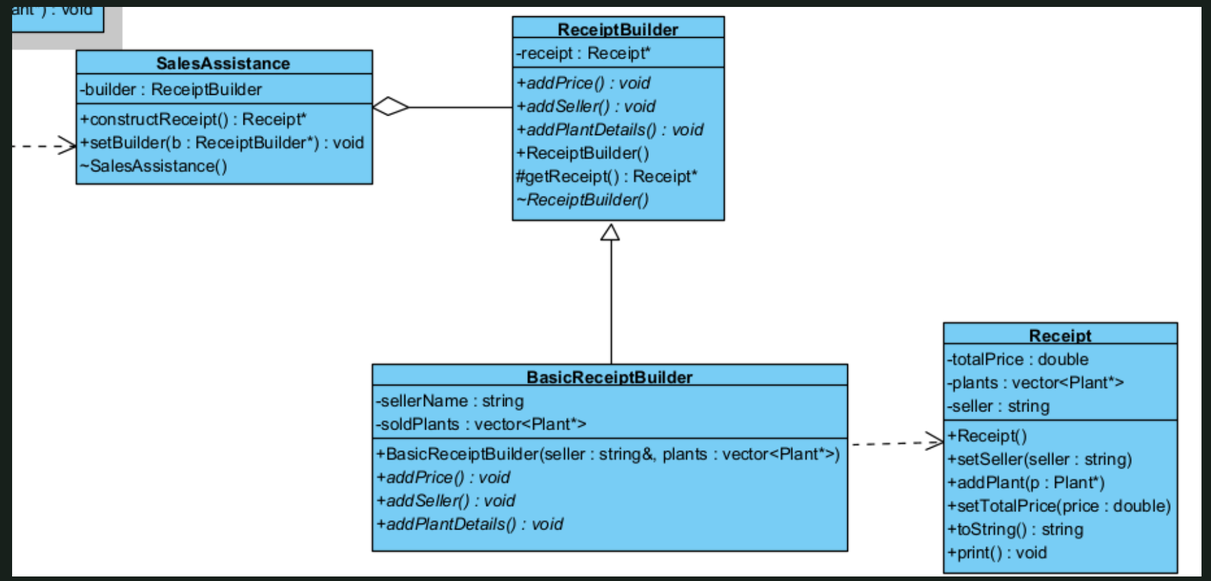


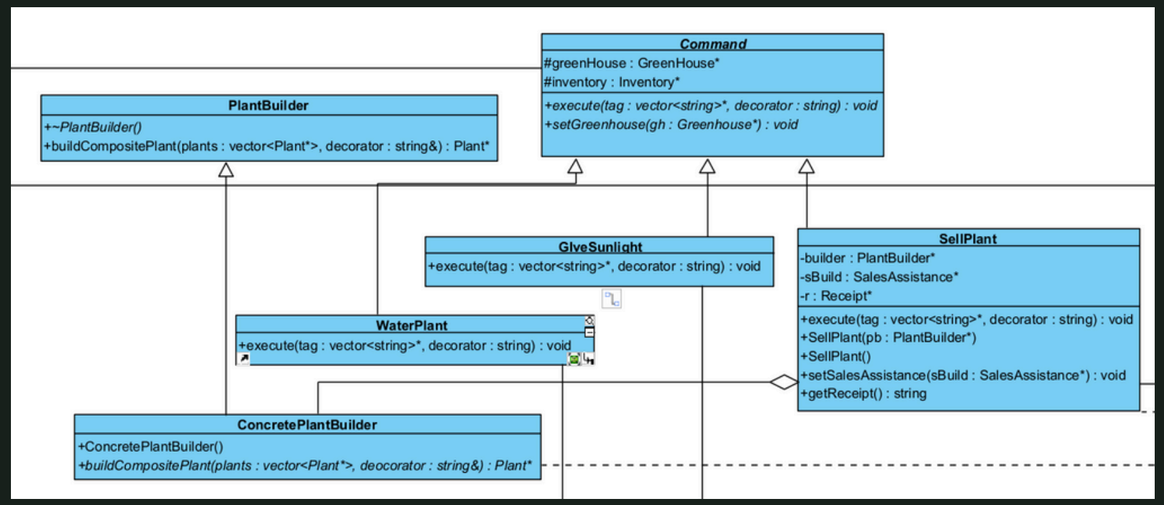
PARTICIPANTS:

Handler: Staff

ConcreteHandler: Manager, SalesClerk and GroundStaff

BUILDER CLASS DIAGRAM





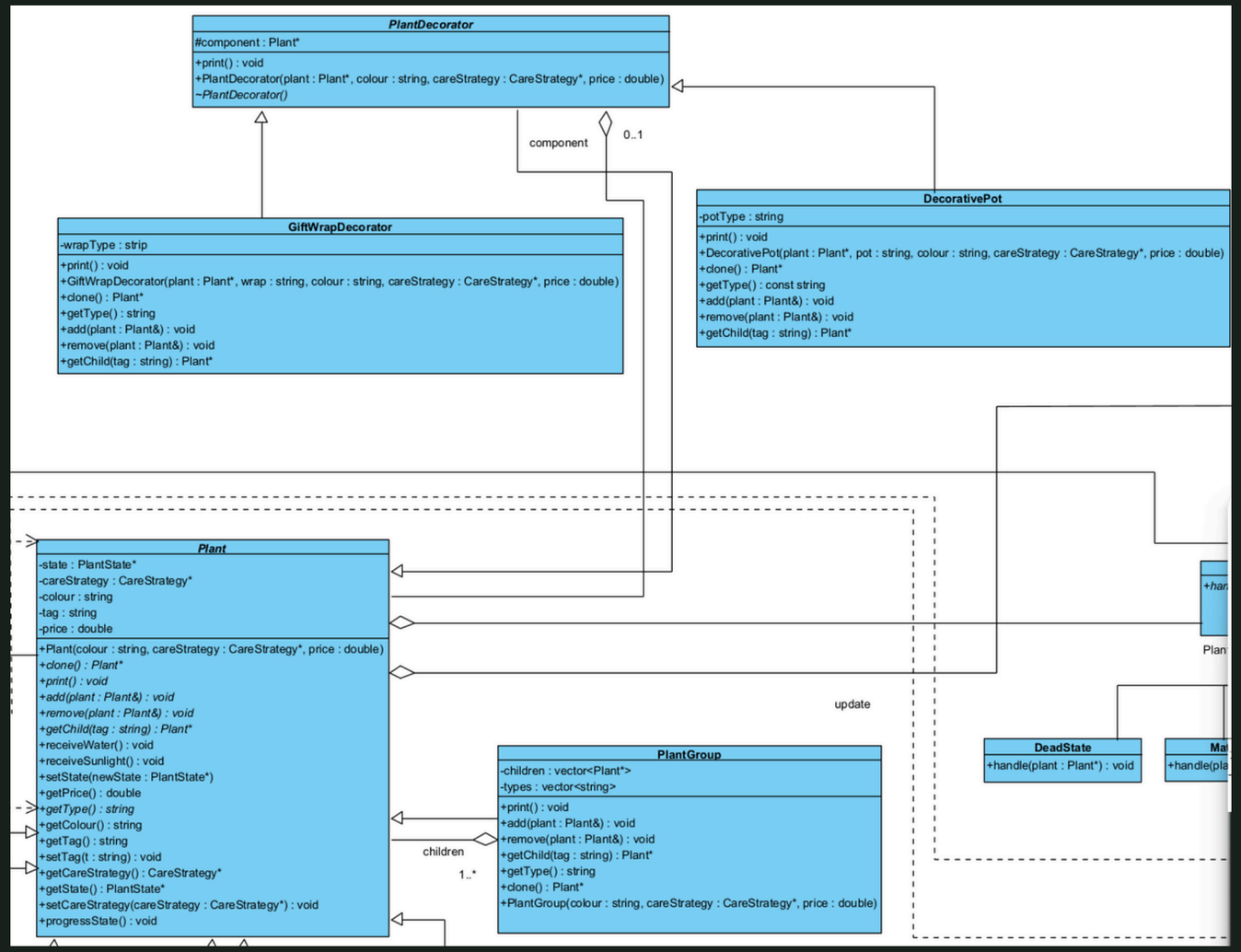
RECEIPT PARTICIPANTS:

Builder: BasicReceiptBuilder
ConcreteBuilder: ReceiptBuilder
Director: SalesAssistance
Product: Receipt

PLANT PARTICIPANTS:

Builder: PlantBuilder
ConcreteBuilder: ConcretePlantBuilder
Director: SellPlant
Product: Plant

DECORATOR CLASS DIAGRAM



PARTICIPANTS:

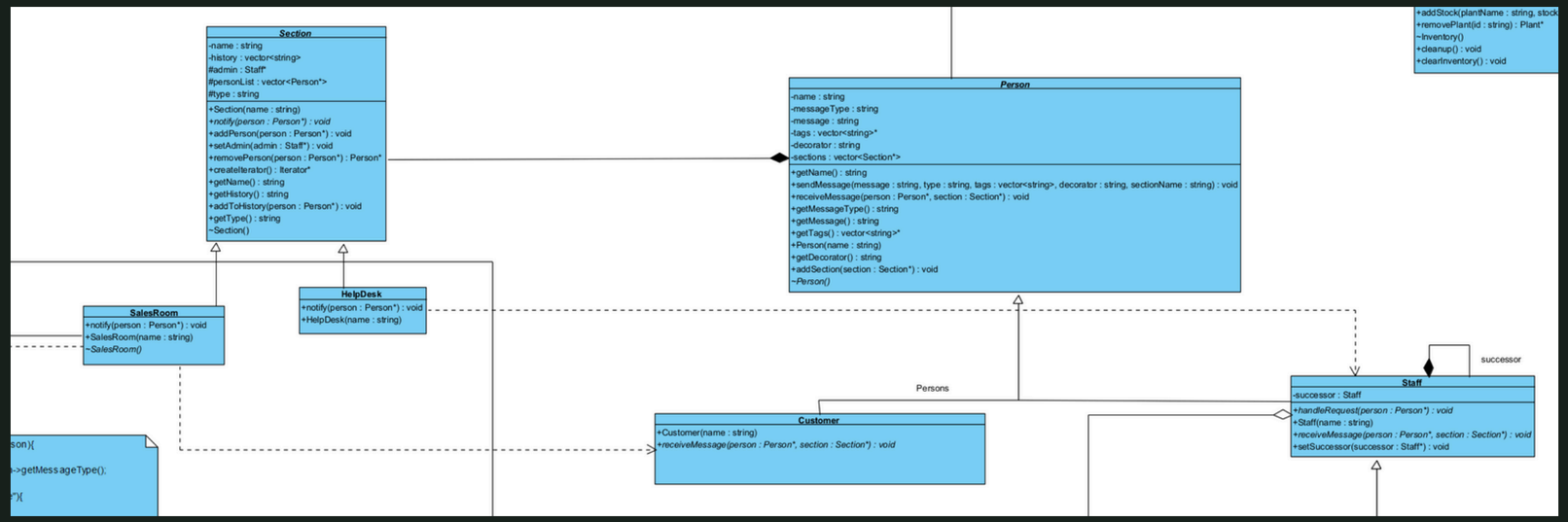
Component: Plant

ConcreteComponent: PlantGroup

Decorator: PlantDecorator

ConcreteDecorator: GiftWrapDecorator and DecorativePot

MEDIATOR CLASS DIAGRAM



PARTICIPANTS:

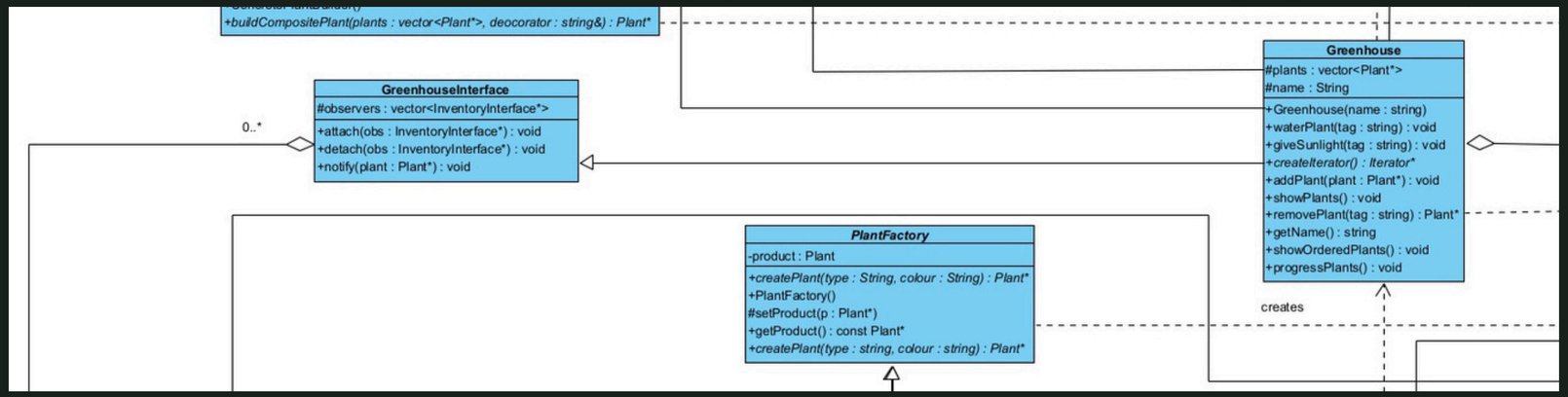
Mediator: Section

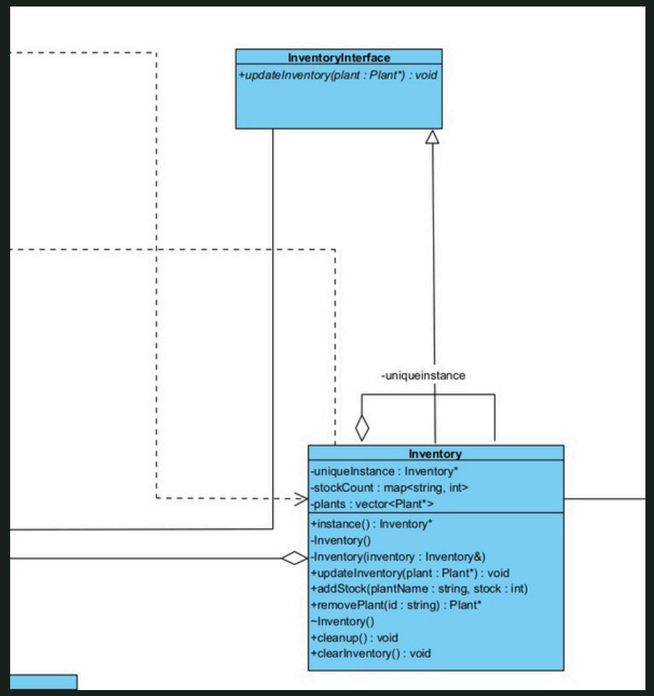
ConcreteMediator: SalesRoom and HelpDesk

Colleague: Person

ConcreteColleague: Customer and Staff

OBSERVER CLASS DIAGRAM





PARTICIPANTS:

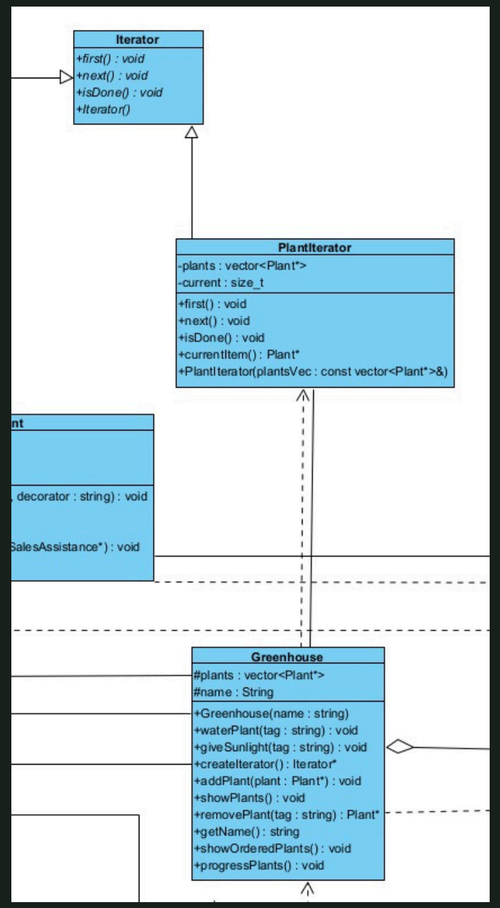
Subject: GreenhouseInterface

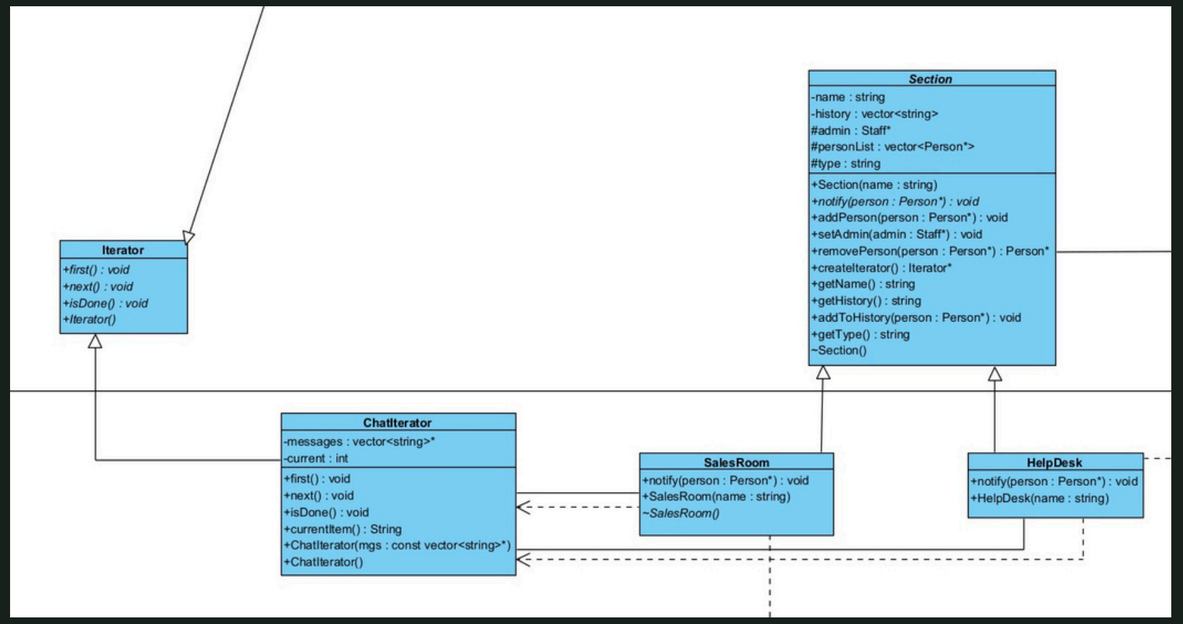
ConcreteSubject: Greenhouse

Observer: Inventory

ConcreteObserver: InventoryInterface

ITERATOR CLASS DIAGRAM





PARTICIPANTS:

Iterator: Iterator

ConcreteIterator: PlantIterator and ChatIterator

Aggregate: Greenhouse and Section

ConcreteAggregate: Greenhouse (for plants) ,and SalesRoom and HelpDesk (for chats)

WALKTHROUGH

UNIT TESTS - FACTORY METHOD

```
// Test FlowerFactory
TEST_F(PlantFactoryTest, FlowerFactoryCreatesRose) {
    Plant* plant = flowerFactory.createPlant("Rose", "Red");
    ASSERT_NE(plant, nullptr);
    EXPECT_EQ(plant->getType(), "Rose");
    EXPECT_EQ(plant->getColour(), "Red");
    EXPECT_DOUBLE_EQ(plant->getPrice(), 15.99);
    delete plant;
}

TEST_F(PlantFactoryTest, FlowerFactoryCreatesTulip) {
    Plant* plant = flowerFactory.createPlant("Tulip", "Yellow");
    ASSERT_NE(plant, nullptr);
    EXPECT_EQ(plant->getType(), "Tulip");
    EXPECT_EQ(plant->getColour(), "Yellow");
    EXPECT_DOUBLE_EQ(plant->getPrice(), 12.49);
    delete plant;
}

TEST_F(PlantFactoryTest, FlowerFactoryInvalidType) {
    Plant* plant = flowerFactory.createPlant("Invalid", "Blue");
    EXPECT_EQ(plant, nullptr);
}

// Test TreeFactory
TEST_F(PlantFactoryTest, TreeFactoryCreatesMango) {
    Plant* plant = treeFactory.createPlant("Mango", "Green");
    ASSERT_NE(plant, nullptr);
    EXPECT_EQ(plant->getType(), "Mango");
    EXPECT_EQ(plant->getColour(), "Green");
    EXPECT_DOUBLE_EQ(plant->getPrice(), 25.99);
    delete plant;
}
```

UNIT TESTS - COMMAND

```
TEST_F(SellPlantTest, Execute_SinglePlant_ReturnsReceipt){
    std::vector<string> tags = {"1"};

    sellCommand->execute(&tags, "wrap");

    std::string receipt = sellCommand->getReceipt();

    EXPECT_FALSE(receipt.empty());
    EXPECT_NE(receipt.find("RECEIPT"), std::string::npos)
        << "Receipt should contain the word 'Receipt'";

}

TEST_F(SellPlantTest, Execute_InvalidTag_NoReceipt){
    std::vector<string> tags = {"999"}; // non-existent tag

    sellCommand->execute(&tags); // perform the sale

    std::string receipt = sellCommand->getReceipt();

    EXPECT_TRUE(receipt.find("No receipt available") != std::string::npos ||
                receipt.empty())
        << "Should not produce a valid receipt for invalid tag";
}


TEST_F(SellPlantTest, Execute_MultiplePlants_GeneratesCompositeReceipt) {
    std::vector<string> tags = {"1", "2"};

    sellCommand->execute(&tags, "pot"); // perform sale with decorator

    std::string receipt = sellCommand->getReceipt();

    EXPECT_NE(receipt.find("PlantA"), std::string::npos);
}
```

UNIT TESTS - DECORATOR

```cpp
// Test DecorativePot
TEST_F(PlantDecoratorTest, DecorativePotWrapsPlant) {
    Plant* decorated = new DecorativePot(rose, "Ceramic", rose->getColour(), nullptr, rose->getPrice());
    rose = nullptr; // Ownership transferred to decorator

    EXPECT_EQ(decorated->getType(), "Rose in Ceramic");
    EXPECT_EQ(decorated->getColour(), "Red");
    EXPECT_DOUBLE_EQ(decorated->getPrice(), 15.99);

    delete decorated;
}

TEST_F(PlantDecoratorTest, DecorativePotPrint) {
    Plant* decorated = new DecorativePot(rose, "Ceramic", rose->getColour(), nullptr, rose->getPrice());
    rose = nullptr; // Ownership transferred to decorator

    testing::internal::CaptureStdout();
    decorated->print();
    std::string output = testing::internal::GetCapturedStdout();
    EXPECT_NE(output.find("Ceramic pot"), std::string::npos);

    delete decorated;
}

// Test GiftWrapDecorator
TEST_F(PlantDecoratorTest, GiftWrapDecoratorWrapsPlant) {
    Plant* decorated = new GiftWrapDecorator(rose, "Ribbon", rose->getColour(), nullptr, rose->getPrice());
    rose = nullptr; // Ownership transferred to decorator

    EXPECT_EQ(decorated->getType(), "Rose with Ribbon");
    EXPECT_EQ(decorated->getColour(), "Red");
    EXPECT_DOUBLE_EQ(decorated->getPrice(), 15.99);

    delete decorated;
}
```

UNIT TESTS - PASSES

```
[----------] 1 test from SectionBasicTest
[ RUN      ] SectionBasicTest.DifferentNamesAndTypes
[       OK ] SectionBasicTest.DifferentNamesAndTypes (0 ms)
[----------] 1 test from SectionBasicTest (0 ms total)

[----------] 4 tests from ConcretePlantBuilderTest
[ RUN      ] ConcretePlantBuilderTest.EmptyInputReturnsNull
[       OK ] ConcretePlantBuilderTest.EmptyInputReturnsNull (0 ms)
[ RUN      ] ConcretePlantBuilderTest.SinglePlantNoDecoratorReturnsSamePointer
[       OK ] ConcretePlantBuilderTest.SinglePlantNoDecoratorReturnsSamePointer (0 ms)
[ RUN      ] ConcretePlantBuilderTest.SinglePlantWithWrapReturnsDecorator
[       OK ] ConcretePlantBuilderTest.SinglePlantWithWrapReturnsDecorator (0 ms)
[ RUN      ] ConcretePlantBuilderTest.MultiplePlantsWithPotReturnsGroupWrapped
[       OK ] ConcretePlantBuilderTest.MultiplePlantsWithPotReturnsGroupWrapped (0 ms)
[----------] 4 tests from ConcretePlantBuilderTest (0 ms total)

[----------] 2 tests from PlantIteratorTest
[ RUN      ] PlantIteratorTest.IteratesCorrectly
[       OK ] PlantIteratorTest.IteratesCorrectly (0 ms)
[ RUN      ] PlantIteratorTest.EmptyVector
[       OK ] PlantIteratorTest.EmptyVector (0 ms)
[----------] 2 tests from PlantIteratorTest (0 ms total)

[----------] 2 tests from ChatIteratorTest
[ RUN      ] ChatIteratorTest.IteratesCorrectly
[       OK ] ChatIteratorTest.IteratesCorrectly (0 ms)
[ RUN      ] ChatIteratorTest.EmptyVector
[       OK ] ChatIteratorTest.EmptyVector (0 ms)
[----------] 2 tests from ChatIteratorTest (0 ms total)

[----------] Global test environment tear-down
[==========] 204 tests from 32 test suites ran. (6 ms total)
[  PASSED  ] 204 tests.
```

MEMORY MANAGEMENT
DEMO MAIN

```
==476691==
==476691== HEAP SUMMARY:
==476691==     in use at exit: 0 bytes in 0 blocks
==476691==   total heap usage: 171 allocs, 171 frees, 83,536 bytes allocated
==476691==
==476691== All heap blocks were freed -- no leaks are possible
==476691==
==476691== For lists of detected and suppressed errors, rerun with: -s
==476691== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

MEMORY MANAGEMENT UNIT TESTS

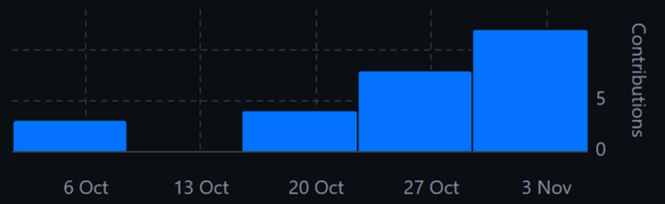
```
[----------] Global test environment tear-down
[==========] 204 tests from 32 test suites ran. (1393 ms total)
[  PASSED  ] 204 tests.
==452495==
==452495== HEAP SUMMARY:
==452495==     in use at exit: 0 bytes in 0 blocks
==452495==   total heap usage: 5,631 allocs, 5,631 frees, 682,768 bytes allocated
==452495==
==452495== All heap blocks were freed -- no leaks are possible
==452495==
==452495== For lists of detected and suppressed errors, rerun with: -s
==452495== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

GITHUB STRATEGY

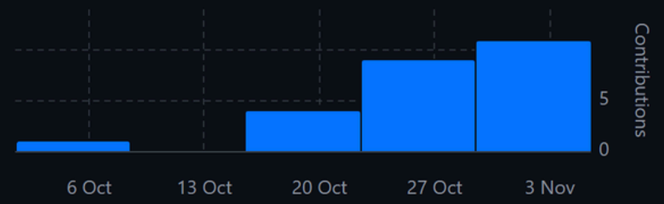
- We made use of the git flow paradigm for version control.
- We had a Dev branch created from main which included all our respective branches, these followed a naming convention of “feature-...” followed by their implementation.
- These branches were then merged back into Dev once all our implementation and testing was done.
- Finally, our Dev branch was merged into main.

GITHUB COMMITS

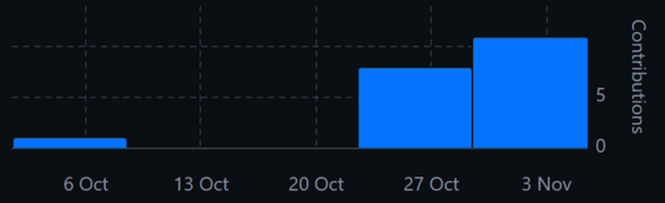
kundaindemera #1
27 commits 148,048 ++ 89,426 --



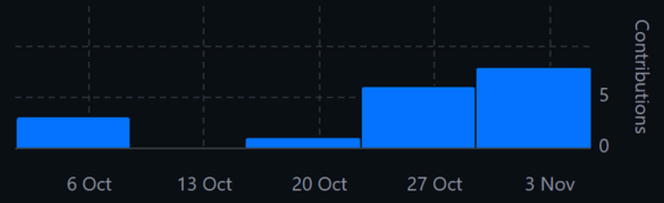
Brayden-ux #2
25 commits 3,096 ++ 489 --



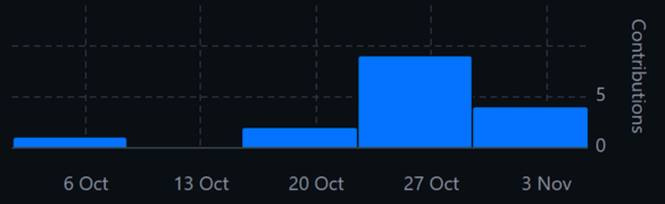
Mosa-L #3
20 commits 1,401 ++ 303 --



SenteMngomez #4
18 commits 5,711 ++ 1,799 --



Padi-le #5
16 commits 569 ++ 192 --



GITHUB PULL REQUESTS

Overview

14 Active pull requests

0 Active issues

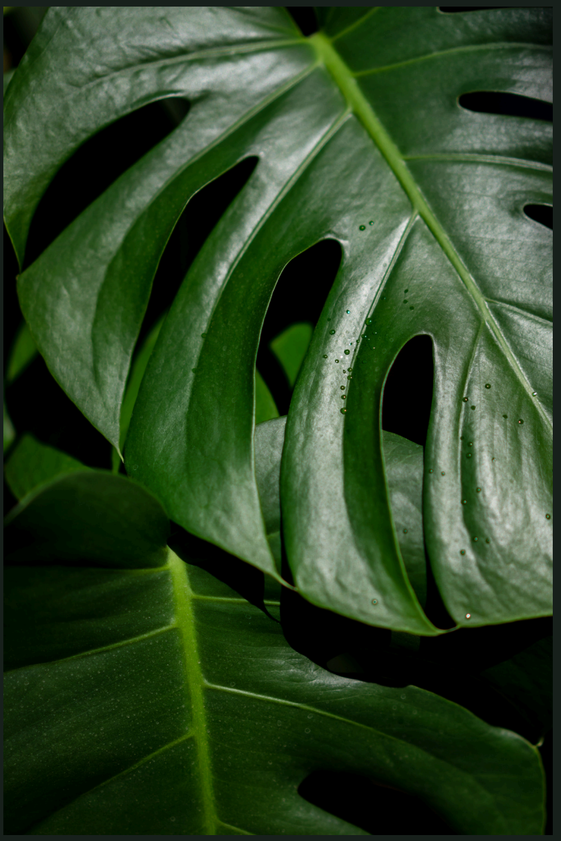
14	0	0	0
Merged pull requests	Open pull requests	Closed issues	New issues

DOXYGEN

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

C **BasicReceiptBuilder**	Minimal implementation of **ReceiptBuilder** for basic receipts
C **Begonia**	Concrete implementation of a **Begonia** plant
C **Cactus**	Concrete implementation of a **Cactus** plant
C **CareStrategy**	Abstract class representing different care strategies for plants
C **ChatIterator**	**Iterator** for chat messages
C **Command**	
C **ConcretePlantBuilder**	Concrete builder that composes multiple **Plant** instances into a single composite **Plant** according to a decorator parameter
C **Customer**	Represents a customer in the system. Inherits from **Person** and implements behaviour for receiving messages sent through a **Section**
C **DeadState**	Represents the dead state of a plant in the state pattern. In this state, the plant can no longer grow or respond to care
C **DecorativePot**	Decorator that adds a decorative pot to a plant
C **FlowerFactory**	Factory for creating flower-type plants
C **GiftWrapDecorator**	Decorator that adds gift wrapping to a plant
C **GiveSunlight**	
C **Greenhouse**	A concrete subject that manages a collection of plants
C **GreenhouseInterface**	An interface for the "Subject" in the Observer design pattern
C **GroundStaff**	Represents ground staff in the system. Inherits from **Staff** and handles "Care" requests by executing watering and sunlight commands on specified plant tags. Also provides helper methods to add plants to a greenhouse and configure care commands
C **HeavyCareStrategy**	Concrete class implementing the **CareStrategy** for plants needing heavy care. This class provides specific implementations for watering, sunlight, and soil care tailored to plants that require more intensive care
C **HelpDesk**	A **HelpDesk** is a concrete **Section** that broadcasts messages to all participants and routes help requests to the admin or the first available **Staff** handler in the section. Behaviour mirrors **Section** derived types such as **SalesRoom** but is intended for assistance
C **Inventory**	A singleton class that manages the inventory of plants in the nursery
C **InventoryInterface**	An interface for the "Observer" in the Observer design pattern
C **Iterator**	



QUESTIONS:)

THANK YOU