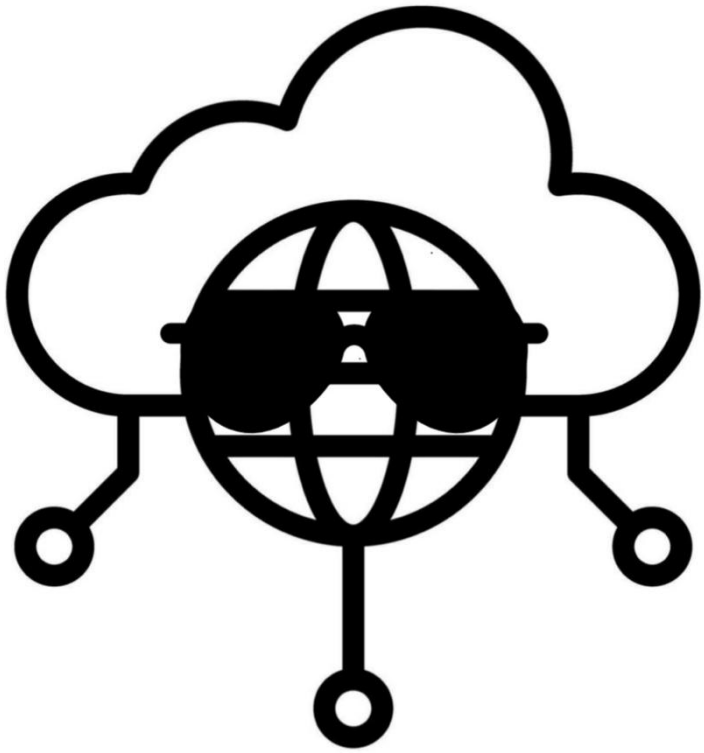


COS214: Final Project System Documentation

THE TEAM : **ANE**

BRIDGET

DAVID

JOSH

KEAGAN

MICHELLE

OPHELLIA

Functional & Non-Functional Requirements

Functional Requirements (FR)

Each design pattern is linked to a clear system behaviour:

No.	Functional Requirement	Design Pattern Used	Description
FR1	The system will use different care strategies depending on plant type.	Strategy	Different strategies (Watering, Sunlight, Fertilizing) define how each plant is cared for.
FR2	The system will allow customers to create and customize plant orders.	Decorator / Factory	Customers can select plant types (created via Factory) and apply custom features (Decorators like Pots, Wrapping, etc.).
FR3	The system will handle staff coordination for tasks like care, watering, and restocking.	Mediator	The NurseryMediator manages communication between Staff, Inventory, and Plants.
FR4	The system will notify customers or staff when inventory or plant status changes.	Observer	Inventory is a Subject, while Staff and Customers act as Observers.
FR5	The system will handle plant purchases using different purchase abstractions (customer vs staff).	Bridge	PurchaseAbstraction bridges the interaction between CustomerPurchase and StaffPurchase through PurchaseImplementor.
FR6	The system will handle multiple types of purchase commands (e.g., PurchaseCommand, CareCommand).	Command	Commands encapsulate actions like processing payments or plant care.

No.	Functional Requirement	Design Pattern Used	Description
FR7	The system will manage customer and staff interactions efficiently.	Mediator	The mediator ensures requests are routed correctly.
FR8	The system allows plants to change states during their life cycle.	State	PlantState changes between SeedlingState, GrowingState, MatureState, and WiltedState.
FR9	The system will track previous purchases and enable undo/restore operations.	Memento	Stores snapshots of purchase or plant status to roll back or review changes.
FR10	The system allows inventory traversal for staff and customers.	Iterator	Enables traversal of plant lists without exposing internal inventory structure.

Non-Functional Requirements (NFR)

No.	Requirement	Quality Attribute
NFR1	The system must support up to 100 plants and 50 users concurrently without performance degradation.	Scalability
NFR2	Changes to plant care strategies should not affect other components.	Maintainability
NFR3	The interface should respond to user commands within 1 second.	Performance
NFR4	The design must allow adding new plant or strategy types without modifying existing code.	Extensibility
NFR5	Doxygen-generated documentation must describe all public classes and methods.	Documentation / Usability

Design Patterns Overview

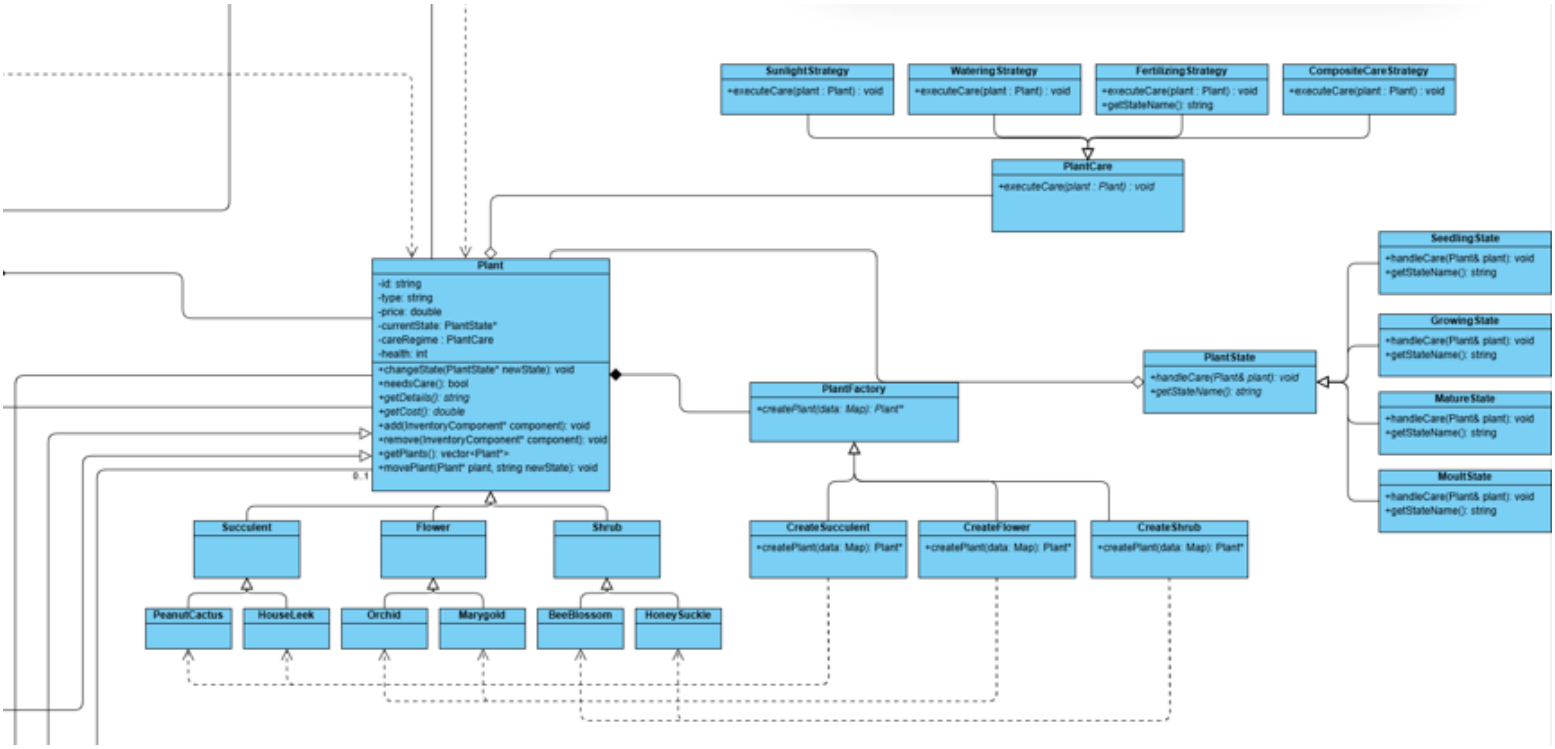
Pattern Type	Pattern	Application in System
Creational	Factory, Builder	Creating plants (PlantFactory), building complex customizations.
Structural	Bridge, Decorator, Composite	PurchaseAbstraction, PlantDecorator, and InventoryComponent manage flexible object structures.
Behavioral	Strategy, State, Command, Observer, Mediator, Memento, Iterator	Governs plant care, plant growth states, communication, undo functionality, and traversals.

TASK 1:

Key Subsystems and Classes

Plant Subsystem (Growth & Care)

Classes: Plant, Succulent, Flower, Shrub, Strategy, PlantState, PlantFactory



1.1) class Plant Hierarchy (Strategy + State + Factory)

Attributes:

private:

```
 string id;
    string type;
    double price;
    string color;
    string sunlight;
    string watering;
    string fertilizer;
    PlantState* state;
    PlantCareStrategy* careStrategy;
```

Public functions :

```
 Plant(string id, string type, double price);
    void executeCarePlan();          // Strategy pattern
    void changeState(PlantState* newState);  // State
transition
    void setCareStrategy(PlantCareStrategy* strategy);
    string getType() const;
    double getPrice() const;
```

Each inherits from Plant:

```
class Succulent :
public Succulent(string id, double price)

class Flower :
public: Flower(string id, double price)

class Shrub :
public: Shrub(string id, double price)
```

```
class Cactus :
public: Cactus(string id, double price)

class Rose :
public: Rose(string id, double price)

class Marigold :
public: Marigold(string id, double price)

class BeeBlossom :
public: BeeBlossom(string id, double price)
```

class PlantCareStrategy

public:

```
    virtual void executeCarePlan(Plant* plant)
    virtual string getStrategyName()
```

Derived Strategies:

class SunlightStrategy

```
public:
void executeCarePlan(Plant* plant) override
string getStrategyName() override
```

class WateringStrategy

```
public:
void executeCarePlan(Plant* plant) override
string getStrategyName() override
```

class FertilizingStrategy

```
public:
 void executeCarePlan(Plant* plant) override
string getStrategyName() override
```

class CompostCareStrategy

```
public:
void executeCarePlan(Plant* plant) override
string getStrategyName() override
```

class PlantState (State Pattern)

public:

```
 virtual void handleCare(Plant* plant)
    virtual string getStateName()
```

Derived States:

class SeedlingState

{ /* handleCare + getStateName */ };

class GrowingState

{ /* handleCare + getStateName */ };

class MatureState

{ /* handleCare + getStateName */ };

class WiltedState

{ /* handleCare + getStateName */ };

class PlantFactory (Abstract Factory)

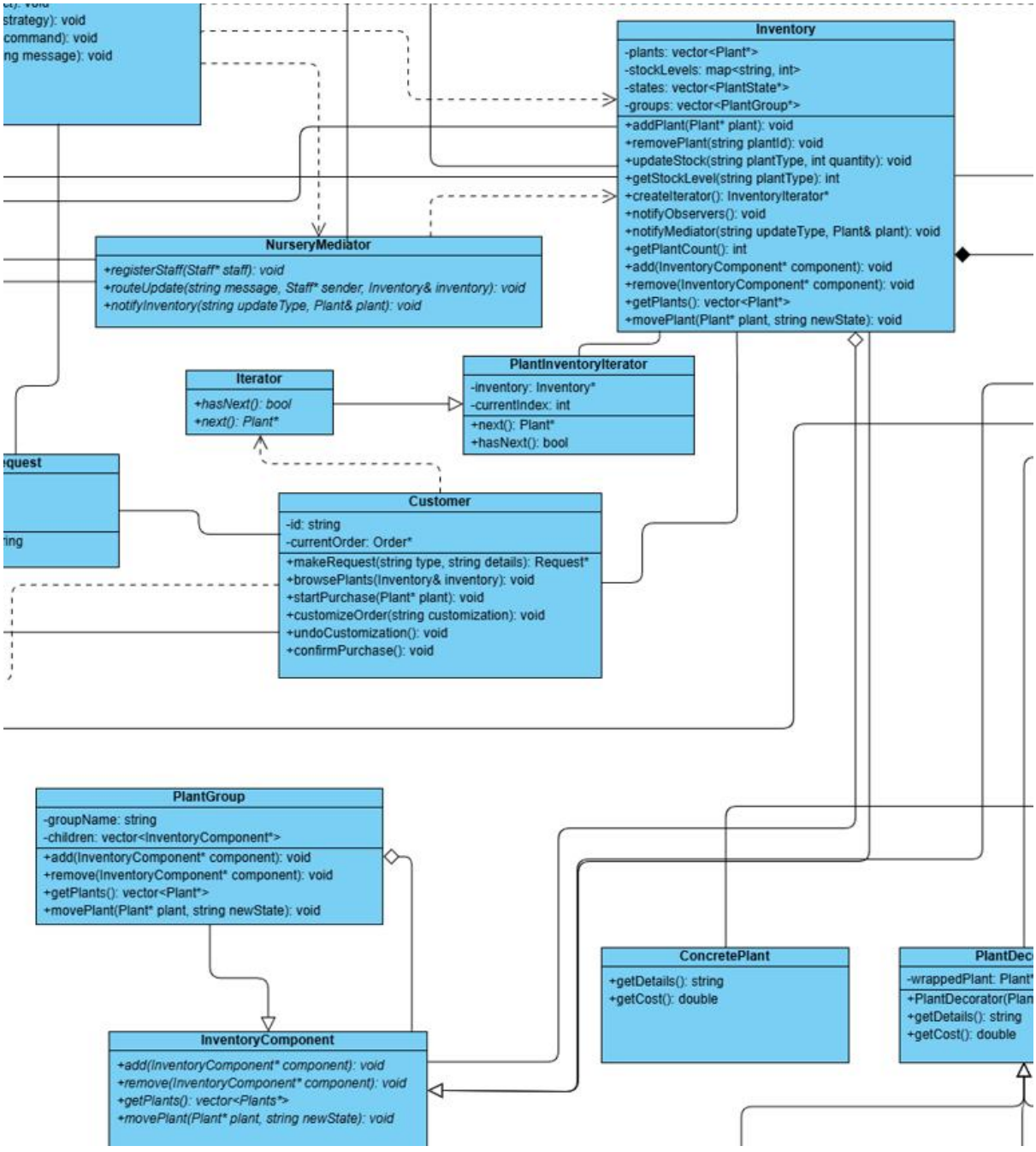
public:

```
 virtual Plant* createPlant(string type) = 0
```

```
class CreateSucculent : public PlantFactory {
```

```
public: Plant* createPlant(string type) override

class CreateFlower : public PlantFactory {
public: Plant* createPlant(string type) override

class CreateShrub : public PlantFactory {
public: Plant* createPlant(string type) override
```

Inventory System (Composite + Iterator + Observer)



```
class InventoryComponent

public:

    virtual void addInventoryComponent(InventoryComponent*
component)

    virtual void removeInventoryComponent(InventoryComponent*
component)

    virtual void showInventory()
```

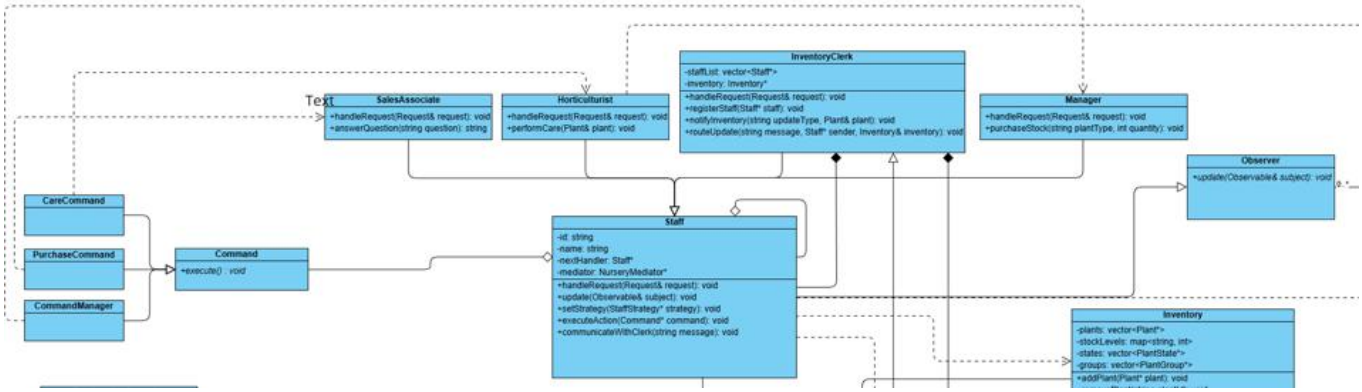
Inventory (Subject + Composite)

```
class Inventory : public InventoryComponent,


public Subject

private:

    vector<Plant*> plants

    string inventoryName

public:

    Inventory(string name)

    void addPlant(Plant* plant)

    void removePlant(string plantId)

    void showInventory() override

    vector<Plant*> getPlants()
```

class InventoryIterator (Iterator Pattern)

```
private:

    vector<Plant*> plants

    int position

public:

    InventoryIterator(vector<Plant*> plants)

    bool hasNext()

    Plant* next()
```

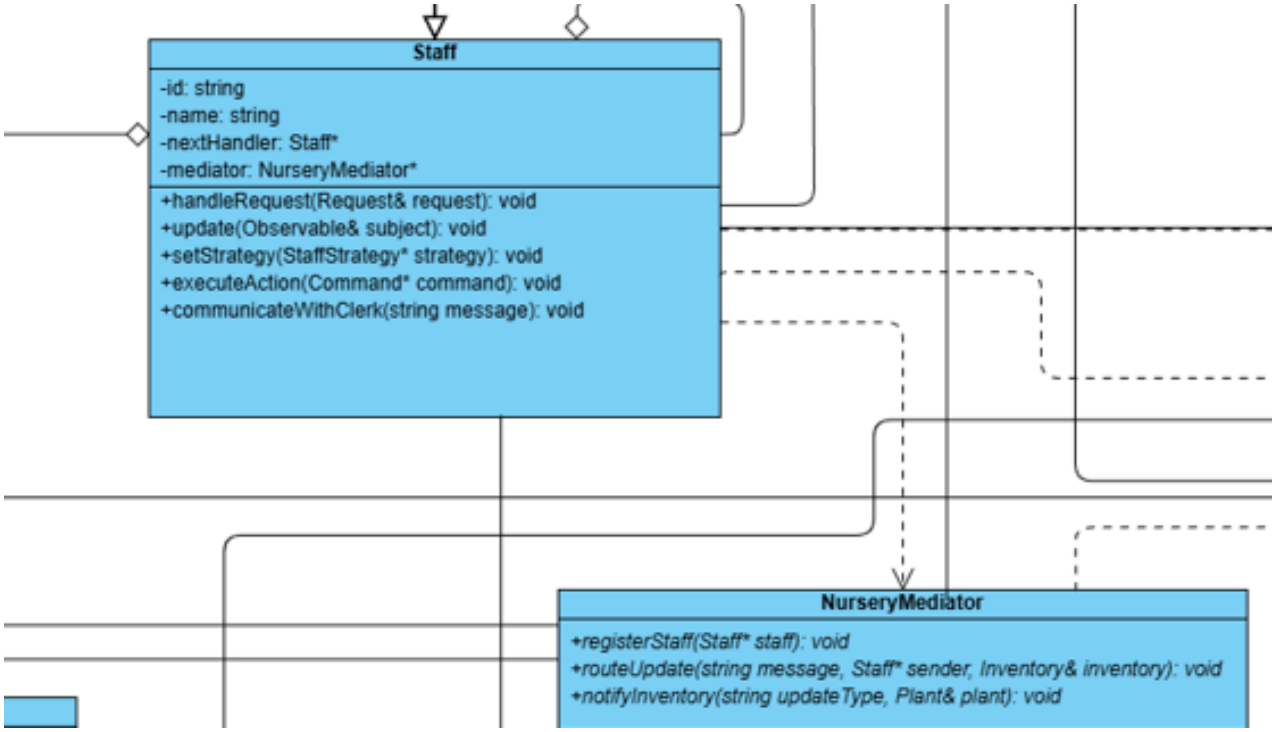
2. Staff System (Mediator + Command + Observer)



Staff

```
class Staff : public Observer {

private:

    string staffID

    string name

    NurseryMediator* mediator

public:

    Staff(string id, string name, NurseryMediator* med)

    void update(Subject* subject) override     // Observer
pattern

    void handleRequest(Request req)
```

class NurseryMediator (Mediator Pattern)



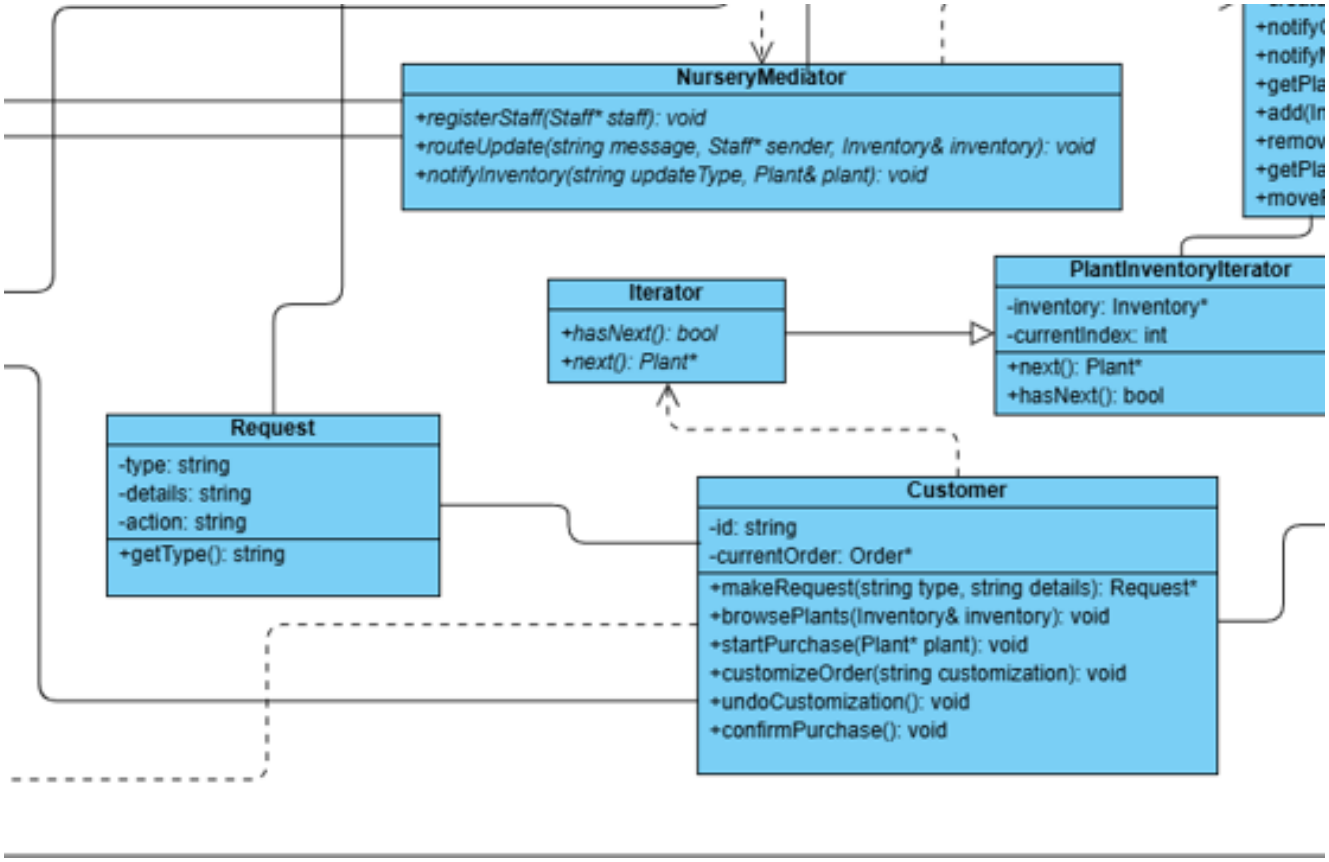

```
private:
    vector<Staff*> staffList
    Inventory* inventory
public:
    NurseryMediator(Inventory* inv)
    void registerStaff(Staff* staff)
    void notifyStaff(Request req)
    void handleInventoryChange(Plant* plant)


class Request
private:
    string type
    string details
    string client
public:
    Request(string t, string d, string c)
```

```
    string getType()
    string getDetails()
    string getClient()
```

Command + Concrete Commands

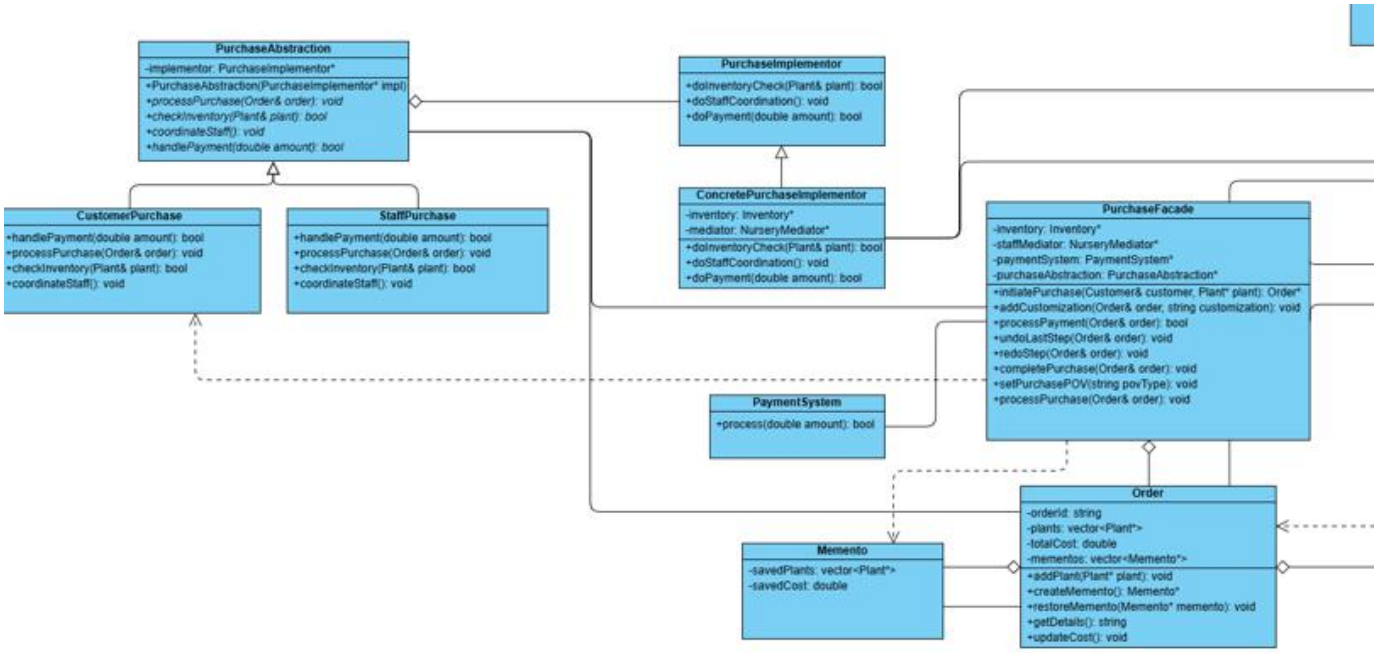
```
class Command
public:
    virtual void execute()

class CareCommand : public Command {
private:
    Staff* staff
    Plant* plant
public:
    CareCommand(Staff* s, Plant* p)
    void execute() override

class PurchaseCommand : public Command {
private:
    Customer* customer
    Plant* plant
public:
    PurchaseCommand(Customer* c, Plant* p)
    void execute() override
```

Customer System (Bridge + Memento + Facade + Decorator)




```
class Customer : public Observer {
private:
    string id
    string name
    vector<Order*> orderHistor
public:
    Customer(string id, string name)
    void update(Subject* subject) override
    void confirmPurchase(Order* order)
    void browsePlants(InventoryIterator it)
```

PurchaseAbstraction (Bridge Pattern)



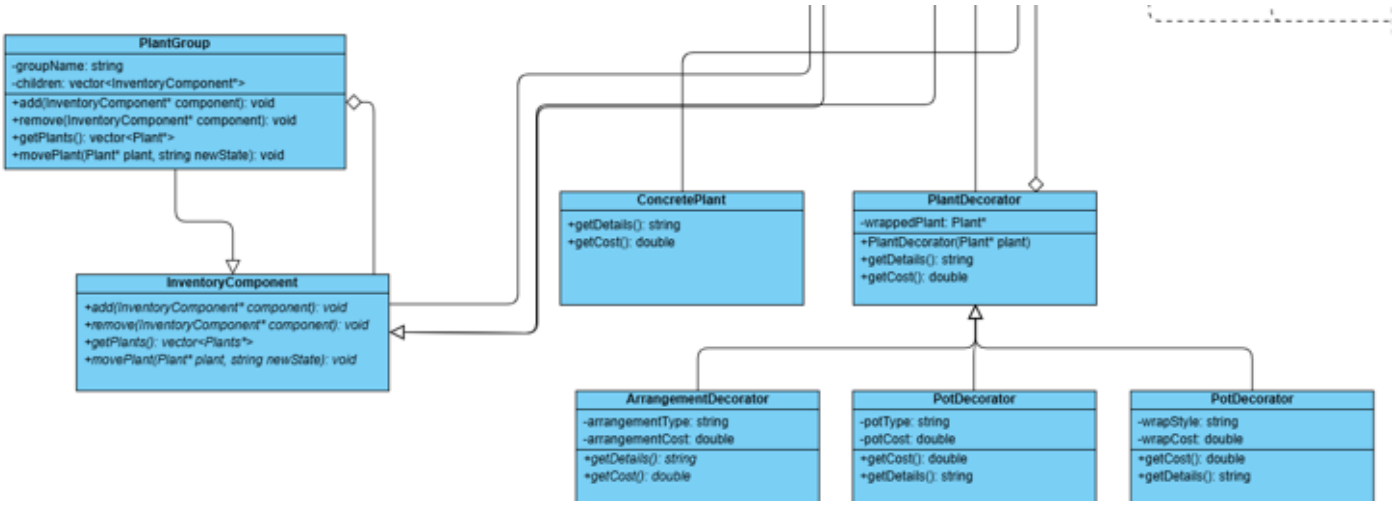

```
class PurchaseAbstraction {
protected:
    PurchaseImplementor* implementor
public:
    PurchaseAbstraction(PurchaseImplementor* impl)
    virtual bool handlePayment(double amount)
```

Derived Classes:

```
class CustomerPurchase : public PurchaseAbstraction {
public:
    CustomerPurchase(PurchaseImplementor* impl)
    bool handlePayment(double amount) override


class StaffPurchase : public PurchaseAbstraction {
public:
    StaffPurchase(PurchaseImplementor* impl)
```

```
    bool handlePayment(double amount) override

class PurchaseImplementor {
public:
    virtual bool processPayment(double amount)

class ConcretePurchaseImplementor : public PurchaseImplementor
{
private:
    PaymentSystem* paymentSystem
public:
    bool processPayment(double amount) override

class PaymentSystem {
public:
    bool process(double amount)

class Memento {
private:
    vector<Plant*> savedPlants
    double savedCost
public:
    Memento(vector<Plant*> plants, double cost)
    vector<Plant*> getSavedPlants()
    double getSavedCost()
```

PlantDecorator (Decorator Pattern)

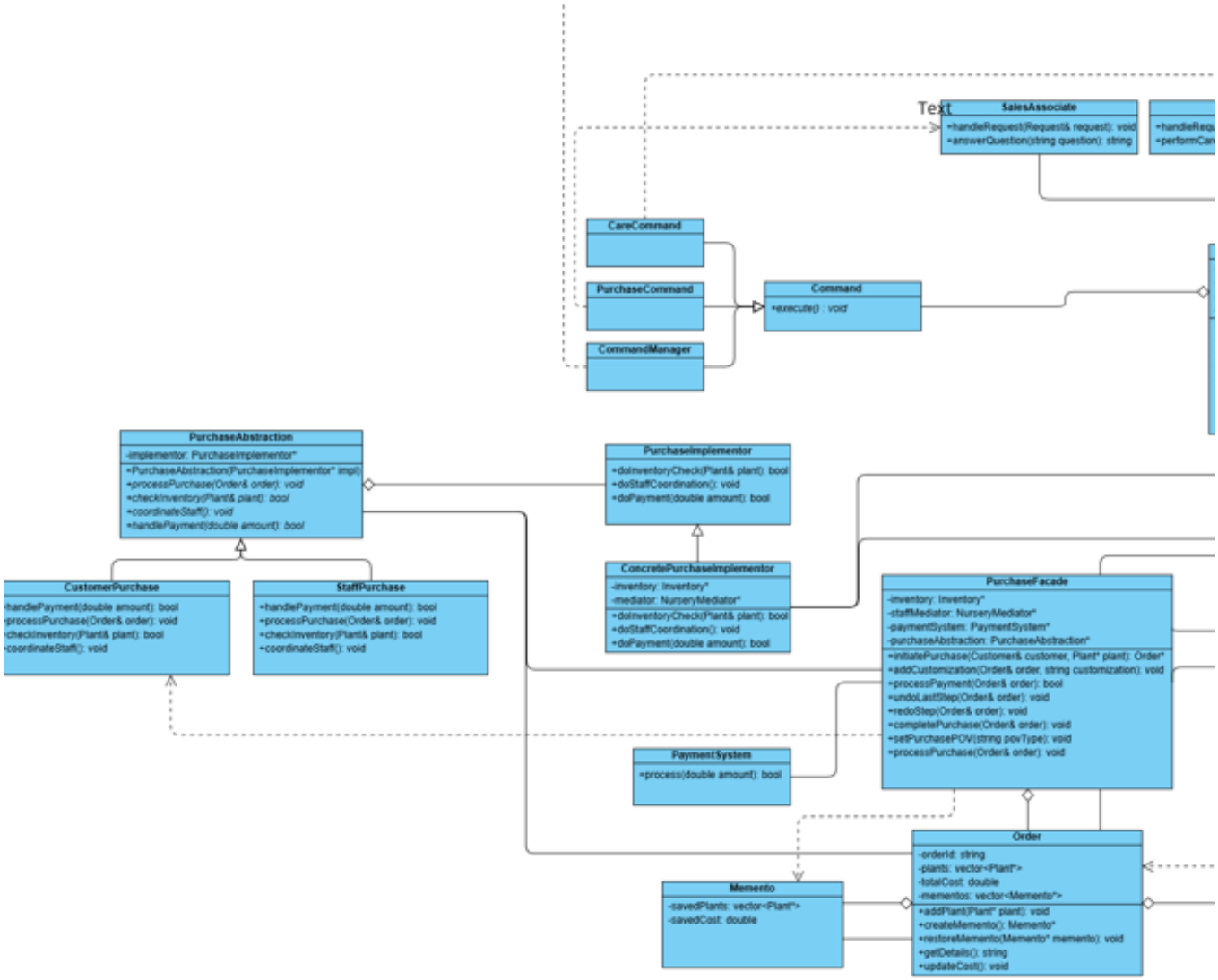



```
class PlantDecorator : public Plant {
protected:
    Plant* component
public:
    PlantDecorator(Plant* plant)
    double getCost() const
```

Derived Decorators:

```
class ArrangementDecorator : public PlantDecorator {
private: double extraCost
public: ArrangementDecorator(Plant* plant, double extra);
double getCost() const override

class PotDecorator : public PlantDecorator {
private: double extraCost
public: PotDecorator(Plant* plant, double extra); double
getCost() const override
```

Auxiliary Classes




```
class Order {
private:
    string orderID
    vector<Plant*> orderedPlants
    double totalCost
    Memento* memento
public:
    Order(string id)
```

```
void addPlant(Plant* plant)
void removePlant(Plant* plant)
double calculateTotal()
Memento* saveState()
void restoreState(Memento* m)
```