Comprehensive Guide to Design Patterns in the Garden Project

This document explains all 13 design patterns used in the Garden Simulation Project. It focuses on three things for each pattern: 1. The core idea (intent and how it works) 2. The main participants and their core methods 3. The relationships between classes and other patterns

1. Factory Method Pattern

Intent: Creates Plant objects without specifying their concrete class. Used for: Generating different plant types such as Flower, Tree, Shrub, and Vegetable.

Participants: - PlantFactory (abstract): Defines createPlant(). - ConcretePlantFactory: Implements createPlant(), returning new instances of plants. - Plant subclasses: Flower, Tree, Shrub, Vegetable.

Core Methods: - createPlant(type, name, species, price): Plant - registerPlantType(type, creator): void

Relationship: Works with Abstract Factory (for bundled creation), State (plants start in SeedlingState), and Strategy (care behavior assigned after creation).

2. Abstract Factory Pattern

Intent: Creates families of related objects (Plant, CareKit, Soil). Used for: Building complete plant configurations with their environment.

Participants: - PlantAbstractFactory: Interface with createPlant(), createCareKit(), createSoil(). - FlowerPlantFactory, TreePlantFactory: Concrete factories. - CareKit, Soil, Plant: Product families.

Relationship: Extends Factory Method to produce grouped, related objects.

3. Singleton Pattern

Intent: Ensures one instance of GardenArea exists to represent the entire nursery. Used for: Centralized management of sections, beds, and plants.

Core Methods: - getInstance(): GardenArea - addSection(section): void - getAllPlants(): List

Relationship: Root structure for Composite pattern; accessed by Facade.

4. Composite Pattern

Intent: Allows the garden to be represented as a tree (sections and beds). Used for: Managing parts (PlantBed) and wholes (GardenSection) uniformly.

Participants: - GardenComponent (abstract): Defines add(), remove(), display(). - GardenSection (composite): Can contain other sections or beds. - PlantBed (leaf): Holds individual plants.

Core Methods: - add(component): void - remove(id): void - display(depth): void

Relationship: Managed by the Singleton GardenArea; supports iteration and reporting.

5. Decorator Pattern

Intent: Dynamically adds features to a Plant without modifying its class. Used for: Adding pots, labels, or gift wraps to plants.

Participants: - Plant (component): Base interface with getDescription(), getPrice(). - PlantDecorator (abstract): Wraps a Plant object and forwards calls. - PottedPlant, LabeledPlant, GiftWrappedPlant: Concrete decorators adding behavior.

Core Methods: - getDescription(): String - getPrice(): double

Relationship: Works with Factory-created plants before sale; complements Command.

6. State Pattern

Intent: Allows a Plant to change behavior as it grows through lifecycle stages. Used for: Managing transitions between Seedling, Growing, Mature, Wilting, Dead states.

Participants: - Plant (context): Holds a reference to current PlantState. - PlantState (interface): Declares water(), fertilize(), grow(), checkTransition(). - Concrete States: SeedlingState, GrowingState, MatureState, WiltingState, DeadState.

Core Methods: - water(Plant): void - fertilize(Plant): void - grow(Plant): void - checkTransition(Plant): void

Relationship: Works with Strategy (care plans), Observer (notifications), and Command (actions).

7. Strategy Pattern

Intent: Defines a family of care algorithms and makes them interchangeable. Used for: Choosing between LowMaintenanceCare, HighMaintenanceCare, and SeasonalCare.

Participants: - CareStrategy (interface): Declares applyCare(), calculateWaterNeeds(). - Concrete Strategies: LowMaintenanceCare, HighMaintenanceCare, SeasonalCare.

Relationship: Used by Plant and interacts with State transitions and Command actions.

8. Command Pattern

Intent: Encapsulates staff actions as command objects. Used for: Watering, Fertilizing, Pruning, Selling, Restocking.

Participants: - Command (interface): execute(), undo(). - Concrete Commands: WaterPlantCommand, FertilizePlantCommand, SellPlantCommand. - CommandInvoker: Executes and stores history.

Relationship: Works with Memento (undo/redo) and affects State and Inventory.

9. Observer Pattern

Intent: Automatically updates observers when a subject (Plant or Inventory) changes. Used for: Monitoring health, growth, and stock levels.

Participants: - Subject (Plant/Inventory): Registers and notifies observers. - Observer (interface): update(). - Concrete Observers: HealthMonitor, GrowthTracker, InventoryObserver.

Relationship: Notified by State changes and supports reporting and iteration.

10. Iterator Pattern

Intent: Provides sequential access to elements in collections without exposing structure. Used for: Traversing Inventory and Garden structures.

Participants: - InventoryIterator (interface): hasNext(), next(). - Concrete Iterators: LowStockIterator, CategoryIterator. - Collection: Inventory.

Relationship: Used by Template Method for reports; linked to Observer.

11. Template Method Pattern

Intent: Defines a report generation algorithm with customizable steps. Used for: Inventory, Sales, and PlantHealth reports.

Participants: - Report (abstract): Defines generateReport(), collectData(), formatBody(). - Concrete Reports: InventoryReport, SalesReport, PlantHealthReport.

Relationship: Uses Iterator for data and Observer for live updates.

12. Facade Pattern

Intent: Provides a simplified interface for controlling the simulation. Used for: Managing plants, staff, inventory, and reports through SimulationFacade.

Participants: - SimulationFacade: Entry point for all major operations. - Subsystems: GardenArea, Inventory, Staff, Reports, Memento.

Relationship: Integrates nearly all other patterns.

13. Memento Pattern

Intent: Saves and restores system state without exposing internal details. Used for: Undo/redo and system backup.

Participants: - SystemOriginator: Creates and restores Mementos. - SystemMemento: Stores state snapshot. - MementoCaretaker: Manages saved states.

Relationship: Works closely with Command and Facade.

Pattern Interaction Summary

1. Factory + Abstract Factory → create and configure plants. 2. State + Strategy → control plant behavior and care response. 3. Command + Memento → perform and undo staff actions. 4. Composite + Singleton → organize the garden layout globally. 5. Decorator → enhance plants dynamically before sale. 6. Observer + Iterator + Template Method → automate monitoring and reporting. 7. Facade → central controller tying all subsystems together.

This structure ensures modular, flexible, and extensible system design, allowing new plant types, actions, and reports to be added without modifying existing code.