Garden Simulation Project
Functional Requirements Specification

Introduction

This document specifies the functional requirements for the Garden Simulation Project, which implements 13 design patterns to create a modular, extensible plant nursery management system.

Plant Management Requirements

Plant Creation

- **FR1.1:** System shall create plants of types: Flower, Tree, Shrub, Vegetable
- **FR1.2:** System shall assign unique names, species, and prices to plants
- **FR1.3:** System shall register new plant types dynamically
- **FR1.4:** System shall create complete plant configurations with `CareKit` and `Soil`

Plant Lifecycle Management

- **FR2.1:** Plants shall progress through states: Seedling → Growing → Mature → Wilting → Dead
- **FR2.2:** System shall automatically transition plants between states based on care and time
- **FR2.3:** Each state shall have different behaviors for `water()`, `fertilize()`, and `grow()` actions
- **FR2.4:** System shall detect when state transitions are needed via `checkTransition()` method

Plant Care System

- **FR3.1:** System shall support multiple care strategies: Low Maintenance, High Maintenance, Seasonal
- **FR3.2:** Each care strategy shall calculate specific water needs via `calculateWaterNeeds()`
- **FR3.3:** System shall allow changing care strategies at runtime
- **FR3.4:** Care strategies shall influence plant state transitions

Garden Structure Requirements

Garden Organization

- **FR4.1:** System shall maintain a single garden instance (Singleton pattern)
- **FR4.2:** Garden shall be organized hierarchically: Sections → Beds → Plants
- **FR4.3:** System shall allow adding/removing garden sections and beds via `add()` and `remove()` methods
- **FR4.4:** System shall display entire garden structure with proper indentation via `display()` method

Plant Placement

- **FR5.1:** System shall assign plants to specific plant beds via `assignPlant()` method
- **FR5.2:** System shall retrieve all plants from the entire garden via `getAllPlants()` method
- **FR5.3:** System shall support nested sections within sections

Sales & Enhancement Requirements

Plant Enhancement

- **FR6.1:** System shall dynamically add features: Potted, Labeled, Gift-Wrapped
- **FR6.2:** Enhancements shall modify plant description and price via `getDescription()` and `getPrice()` methods
- **FR6.3:** System shall allow multiple enhancements on single plants
- **FR6.4:** Enhanced plants shall maintain original plant functionality

Sales Operations

- **FR7.1:** System shall execute plant sales transactions via `SellPlantCommand`
- **FR7.2:** System shall support undo/redo of sales operations
- **FR7.3:** Sales shall update inventory automatically

Staff & Action Requirements

Staff Actions

- **FR8.1:** System shall execute staff commands: Water, Fertilize, Prune, Sell, Restock
- **FR8.2:** All commands shall support undo/redo functionality

- **FR8.3:** Command history shall be maintained for rollback
- **FR8.4:** Commands shall affect plant states and inventory

Action Management

- **FR9.1:** System shall encapsulate each action as a command object
- **FR9.2:** Command execution shall be decoupled from invocation
- **FR9.3:** System shall support macro commands (multiple actions)

Monitoring & Reporting Requirements

Observation System

- **FR10.1:** System shall notify observers of plant state changes
- **FR10.2:** System shall notify observers of inventory changes
- **FR10.3:** Observers shall include: `HealthMonitor`, `GrowthTracker`, `InventoryObserver`
- **FR10.4:** System shall support adding/removing observers dynamically via `registerObserver()` and `removeObserver()`

Reporting System

- **FR11.1:** System shall generate: Inventory Reports, Sales Reports, Plant Health Reports
- **FR11.2:** All reports shall follow standardized generation algorithm via `generateReport()` template method
- **FR11.3:** Reports shall use iterators for data collection in `collectData()` method
- **FR11.4:** System shall support custom report formats via `formatBody()` method

Data Access

- **FR12.1:** System shall provide iterators for: Low Stock items, Category-based items
- **FR12.2:** Iterators shall hide internal collection structures via `hasNext()` and `next()` methods
- **FR12.3:** System shall support multiple iteration strategies

System Management Requirements

State Persistence

- **FR13.1:** System shall save complete garden state as mementos via `createMemento()` method
- **FR13.2:** System shall restore garden state from mementos via `restoreMemento()` method
- **FR13.3:** Mementos shall encapsulate state without exposing internals
- **FR13.4:** System shall manage memento history via `MementoCaretaker`

System Control

- **FR14.1:** System shall provide simplified facade for all major operations via `SimulationFacade`
- **FR14.2:** Facade shall integrate: Plant management, Staff actions, Inventory, Reporting
- **FR14.3:** System shall handle all pattern coordination internally

Integration Requirements

Pattern Coordination

- **FR15.1:** Factory-created plants shall automatically receive initial state and strategy
- **FR15.2:** State changes shall trigger observer notifications
- **FR15.3:** Commands shall work with mementos for undo/redo functionality
- **FR15.4:** Reports shall use iterators and observer data for real-time reporting

Extensibility

- **FR16.1:** System shall allow adding new plant types without modifying existing code
- **FR16.2:** System shall support new care strategies dynamically
- **FR16.3:** System shall allow new report types through inheritance
- **FR16.4:** System shall support new commands without structural changes