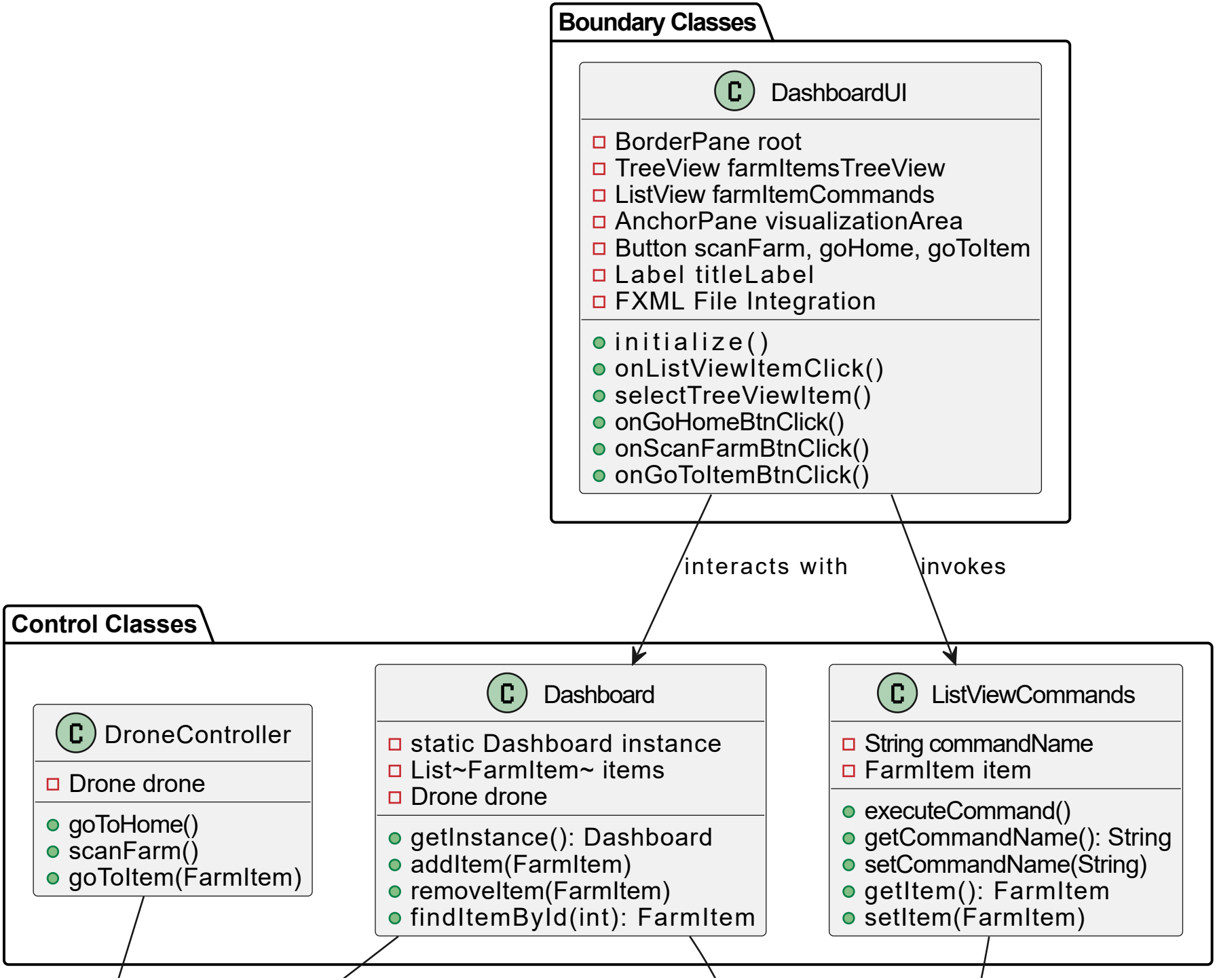
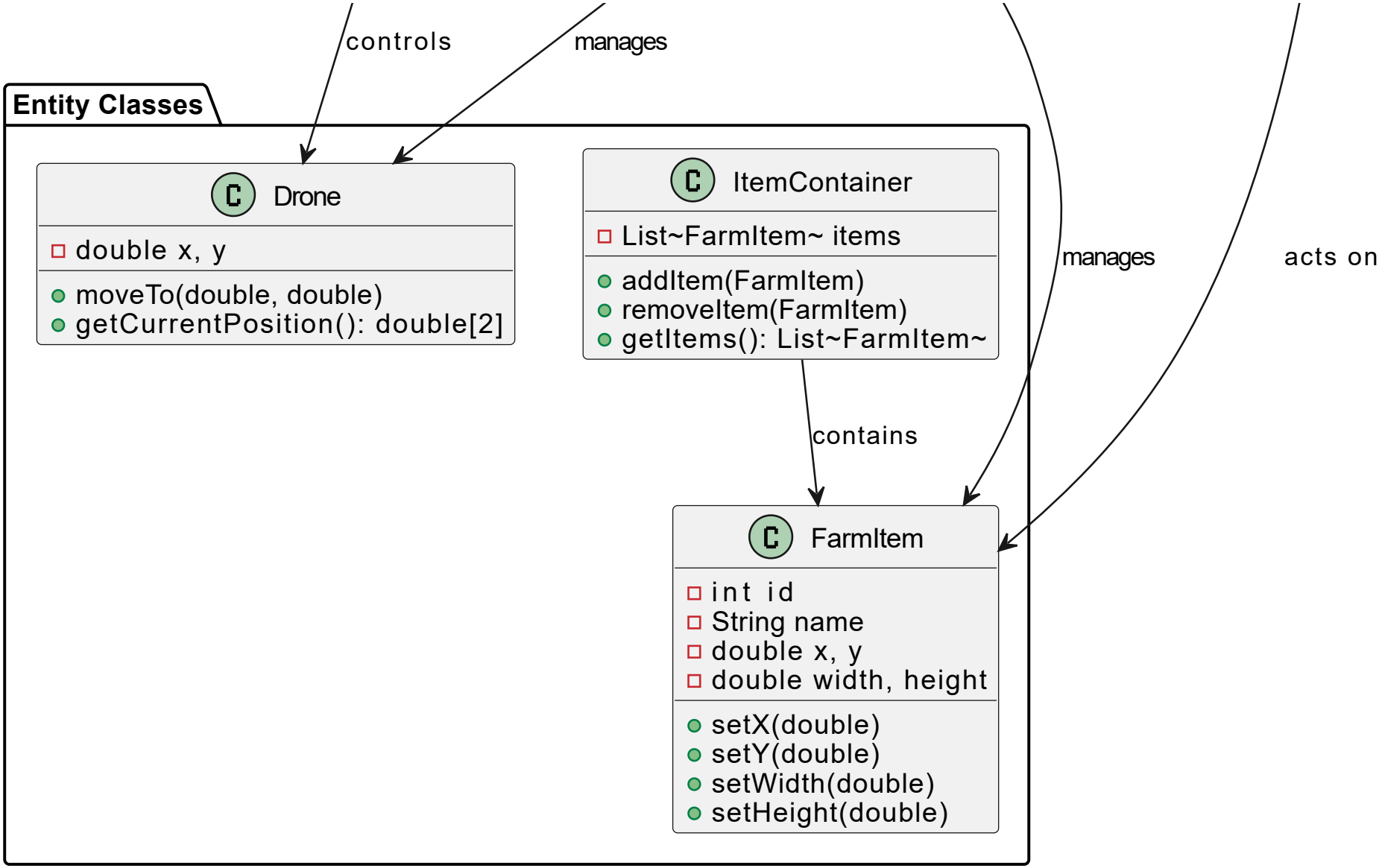

Boundary Classes
DashboardUI
BorderPane root
TreeView farmItemsTreeView
ListView farmItemCommands
AnchorPane visualizationArea
Button scanFarm, goHome, goToItem
Label titleLabel
FXML File Integration
initialize()
onListViewItemClick()
selectTreeViewItem()
onGoHomeBtnClick()
onScanFarmBtnClick()
onGoToItemBtnClick()
interacts with
invokes
Control Classes
DroneController
Drone drone
goToHome()
scanFarm()
goToItem(FarmItem)
Dashboard
static Dashboard instance
List~FarmItem~ items
Drone drone
getInstance(): Dashboard
addItem(FarmItem)
removeItem(FarmItem)
findItemById(int): FarmItem
ListViewCommands
String commandName
FarmItem item
executeCommand()
getCommandName(): String
setCommandName(String)
getItem(): FarmItem
setItem(FarmItem)


controls
manages
Entity Classes
Drone
double x, y
moveTo(double, double)
getCurrentPosition(): double[2]
ItemContainer
List~FarmItem~ items
addItem(FarmItem)
removeItem(FarmItem)
getItems(): List~FarmItem~
contains
manages
acts on
FarmItem
int id
String name
double x, y
double width, height
setX(double)
setY(double)
setWidth(double)
setHeight(double)

Class Diagram Description

Boundary Classes-

a.DashboardUI- It is a representation of the dashboard's user interface, which includes buttons for operations related to drones, a list view for commands, and a tree view for agricultural objects. It offers ways to deal with user input such button pushes and clicks.

Control Classes-

a. Dashboard- It is a singleton class that controls the system's general state, which includes a drone and a list of farm objects. It communicates with the user interface for updates and offers ways to add, remove, and identify agricultural things.

b. DroneController- The drone controller manages the drone's functions, including returning home, farm scanning, and locating particular farm objects. It connects the dots between the drone's physical motions and the actions of the user interface.

c.ListViewCommands- The individual commands connected to agricultural objects are represented by ListViewCommands. It provides ways to maintain command details, such as the name and related farm item, and to execute commands.

Entity Classes-

a. FarmItem- Represents a farm item with properties such as dimensions, coordinates, name, and ID. It has ways to change its size and position.

b. ItemContainer- Manages a collection of farm items, allowing items to be added, removed, or retrieved as a list. It serves as a central repository for farm items in the system.

c. Drone- Models the drone's current state, including its position. It includes methods to move the drone to specified coordinates and retrieve its current position.

How Singleton and Composite Patterns Are Used in the Farm Management System

In order to provide a modular, scalable, and effective system for managing farm resources and tasks, the Farm Management System combines the Singleton and Composite Design Patterns. The project description and class diagram that are presented make it clear that these patterns are essential to the architecture of the system.

Singleton Pattern in Farm Management System

The Singleton Pattern is implemented in the Dashboard class to ensure a single, globally accessible instance of the dashboard controller.

Why Singleton for Dashboard?

- The Dashboard serves as the system's central hub, managing:
    - A Drone that scans the farm and performs navigation tasks.
    - A List of Farm Items representing all resources in the farm.
- Using a singleton ensures that:
    - There is a single source of truth for managing the farm's state.
    - Inconsistencies are avoided by preventing the creation of multiple dashboard instances.

Implementation Details:

1. **Class Details**:
    - To hold the single instance, the Dashboard class has a static Dashboard instance variable.
    - Multiple instances cannot be created because of a private constructor.
    - The single instance can be accessed under control using the getInstance() method.

2. **Key Functionalities in Singleton**:
    - To handle farm items, perform addItem(FarmItem) and removeItem(FarmItem).
    - using the DroneController to communicate with the drone for navigation and farm scanning operations.

Example Workflow:
The Dashboard singleton interacts with the DroneController to start scanning when a user selects "Scan Farm" on the DashboardUI, maintaining a consistent system state throughout the process.

Composite Pattern in Farm Management System

The Composite Pattern is applied to manage hierarchical structures of farm items, where items can either be individual entities (e.g., a cow) or containers (e.g., a barn that holds multiple cows).

Why Composite for Farm Items?

- Farms have a hierarchical structure:
    - An ItemContainer like a barn can hold multiple items (e.g., cows or tools).
    - A FarmItem can either be a simple item or an item container.
- The composite pattern allows:
    - Treating individual items and groups of items uniformly.
    - Recursive operations on the hierarchy, such as adding, deleting, or retrieving items.

Implementation Details:

1. **Class Hierarchy**:
    - FarmItem (Abstract Component): Defines the common properties (e.g., id, name, dimensions) and behaviors for all farm items.
    - ItemContainer (Composite): Extends FarmItem and maintains a list of child FarmItem objects. Provides methods like addItem() and removeItem().
    - SimpleFarmItem (Leaf): Represents individual farm items without children. Operations like addItem() throw an UnsupportedOperationException.
2. **Key Functionalities in Composite**:
    - The ItemContainer enables hierarchical organization of farm resources.
    - The drone can navigate to any item or container using the structure provided by the composite.

Example Workflow:

- When a user adds a "Barn" using the DashboardUI, an ItemContainer is created and added to the visualization pane.

- Additional items (e.g., cows or tools) can be added to the barn using the addItem() method.
- The "Go To Item" feature uses the hierarchy to navigate the drone to a specific item or container.

How These Patterns Work Together

1. **Centralized Control with Singleton**:
    - The Dashboard Singleton ensures there is a consistent control point for managing all operations, including item hierarchy updates and drone navigation.
2. **Hierarchical Management with Composite**:
    - The Composite Pattern organizes farm items in a way that the Dashboard Singleton can efficiently process and manage operations like adding, removing, and locating items.
3. **Interaction Example**:
    - When "Scan Farm" is clicked, the Dashboard Singleton retrieves all farm items through the composite structure (ItemContainer) and commands the drone to navigate or monitor specific items.

Summary

- The Singleton Pattern ensures the Dashboard remains the single source of truth, managing the overall system state and interactions.
- The Composite Pattern enables hierarchical organization of farm items, simplifying their management and drone navigation.
- Together, these patterns make the Farm Management System scalable, user-friendly, and efficient in modernizing traditional farming methods.