SANTA CLARA UNIVERSITY
SCHOOL OF ENGINEERING

Automated Computerized Gardening System:

By:
Group 1:
Divyanth Chalicham
Varshit Purna Amrutham
Wineel Wilson Dasari

Introduction :

The Computerized Garden Simulator is a comprehensive Java application that creates a virtual garden management experience. Users can interact with the simulation through a graphical JavaFX interface. The software enables automated environmental control, including plant watering, temperature regulation, and pest management. Its architecture incorporates multiple controllers to handle different simulation components, and it implements robust logging capabilities using the Log4j framework.

Project Statement:

Managing large, diverse gardens is labor-intensive and prone to errors, requiring consistent irrigation, temperature control, pest management, and activity monitoring. Traditional methods are inefficient and unsustainable for complex ecosystems.
The solution is a Computerized Gardening System (CGS) that automates these tasks, integrates multiple subsystems, and provides detailed logs, ensuring efficient garden maintenance with minimal manual effort.

Project Objective:

The main objectives of this project are:

1) To develop a realistic simulation of garden maintenance.
2) Developing a multi-interface system that enables user interaction via command-line and graphical platforms.
3) To manage and control different environmental factors such as rain, temperature, and pest control.

4) To provide a flexible and extensible architecture for adding new features and plant types.

5) To implement detailed logging for tracking the simulation process.

Project Scope:

This project offers a digital garden management system that includes garden activities and environmental factors.

1. Simulating rainfall and sprinkler activation.
2. Simulating pest attacks and applying pesticides.
3. Managing different types of plants with specific requirements and susceptibilities
4. Adjusting garden temperature with a heating system.
5. Providing a GUI for user interaction with the garden simulation.

Methodology:

Tools, Technologies, and Frameworks Used

1. Programming Language:
    a. Java: Core programming language for implementing the system.
2. User Interface:
    a. JavaFX: For developing a responsive, user-friendly graphical interface.
3. Development Tools:
    a. IntelliJ IDEA / Eclipse: IDEs for efficient coding and debugging.
4. Design Tools:
    a. Draw.io: For creating UML diagrams like class diagrams, sequence diagrams, and activity diagrams.

System Architecture:

The system architecture of the Automated Garden Simulator is based on a modular design
The main components include:

1. Simulation: Handles the lifecycle and timing of the garden simulation.
2. Controllers: Manage specific aspects of the garden simulation.
3. Models: Represent different types of plants with specific attributes.
4. GUI: Provides a graphical interface for user interaction.

Workflow:

1. Requirements Analysis
    a. Gather requirements from gardening resources, gardeners, and online research.

   b. Identify key features (e.g., watering, pest control, heating).
2. Design Phase
   a. Create UML diagrams (Class Diagram, Sequence Diagram, Use Case Diagram).
3. Implementation Phase
   a. Develop modules in Java.
   b. Build a GUI using JavaFX.
4. Testing
   a. Unit testing for individual modules.
   b. Integration testing to ensure all components work together.
   c. Stress testing for prolonged operation.
5. Deployment
   a. Package the system for use.
   b. Prepare the help manual and provide detailed documentation.

Class Descriptions:

com.garden.controllers :-

1. **GardenController**: Manages overall garden operations and coordinates interactions between different controllers.
2. **RainController**: Simulates rainfall and activates the sprinkler system if rainfall is insufficient.
3. **TemperatureController**: Adjusts the temperature of the garden environment and activates heating if necessary.
4. **PestAttackController**: Simulates pest attacks on plants and determines the impact on plants based on their susceptibility and pesticide status.
5. **PesticideController**: Applies pesticides to plants to enhance resistance against pests.
6. **SprinklerController**: Manages the sprinkler system and calculates and distributes water based on plant requirements.
7. **HeatingController**: Manages the heating system to ensure temperatures stay above a minimum safe threshold.

com.garden.models :-

1. **Plant**: Abstract base class representing a generic plant, with properties like water requirements, temperature tolerance, and pests.
2. **Rose**: Specific type of plant with specific water requirements and pests.
3. **Orange**: Specific type of plant with specific water requirements and pests.

4. **Tomato**: Specific type of plant with specific water requirements and pests.

com.garden.simulation :-

1. **GardenSimulator:** Main entry point for the CLI version of the simulation, managing the simulation lifecycle and daily operations.
2. **GardenSimulatorAPI:** Provides an API for interacting with the garden simulation, initializing the garden and allowing manipulation of simulation parameters.
3. **GardenThread:** Custom thread class for handling asynchronous garden tasks.
4. **MyTimer:** Manages simulation time, tracking days and hours.

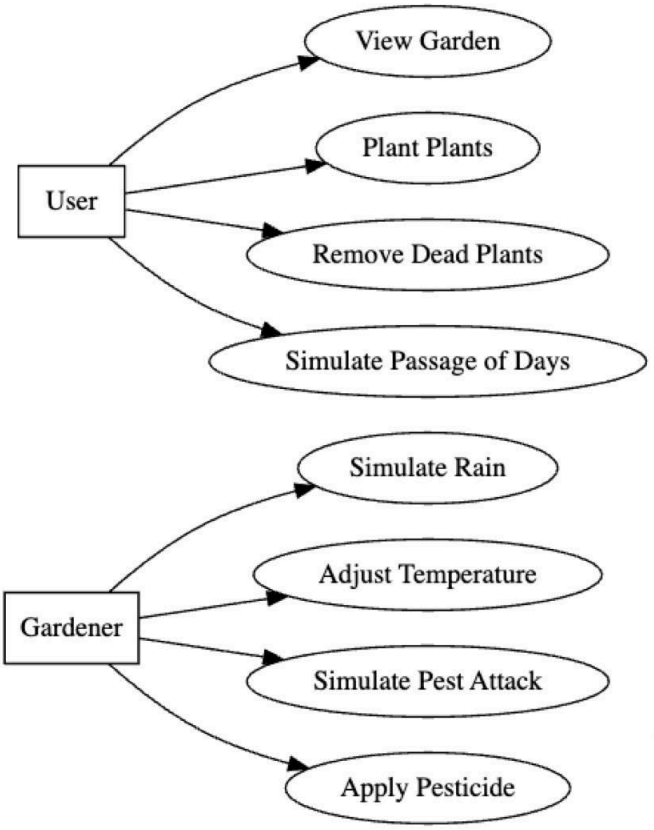
com.garden.gui :-

1. **GUIMain:** Main entry point for the JavaFX application, loading the GUI and starting the application.
2. **ViewController:** Handles GUI interactions and connects user actions with simulation operations.
3. **hello-view.fxml:** FXML layout file for the GUI.

SANTA CLARA UNIVERSITY
SCHOOL OF ENGINEERING

UML DIAGRAMS :

Use Diagram:-



Description:-

The use case diagram illustrates the interactions between users (actors) and the system, highlighting the key functionalities that the system offers.

Actors:-

1. **User**: The primary actor who interacts with the simulation via the GUI or CLI.
2. **Gardener**: An internal actor that manages the garden operations and ensures the health of the plants.

Use Cases:-

1. **Initialize Garden:** The user starts the garden simulation.
2. **Simulate Rain:** The user simulates rain in the garden.
3. **Adjust Temperature:** The user adjusts the temperature of the garden.
4. **Simulate Pest Attack:** The user simulates a pest attack on the plants.
5. **Apply Pesticide:** The gardener applies pesticides to the plants.
6. **Activate Sprinklers:** The gardener activates the sprinkler system when necessary.
7. **Activate Heating:** The gardener activates heating when the temperature is too low.
8. **Update Plant Status:** The gardener updates the status of each plant based on the conditions.
9. **Log Events:** The system logs significant events and actions.

Use Case Descriptions:-

Initialize Garden:

1. **Actor**: User
2. **Description**: The user initializes the garden, which loads the plant configurations and sets up the initial state.
3. **Precondition**: Configuration file is available.
4. **Postcondition**: Garden is initialized with plants.

Simulate Rain:

1. **Actor**: User
2. **Description**: The user simulates rain in the garden. If the rainfall amount is insufficient, the gardener activates sprinklers.
3. **Precondition**: Garden is initialized.
4. **Postcondition**: Plants receive water from rain or sprinklers.

Adjust Temperature:

1. **Actor**: User
2. **Description**: The user adjusts the garden temperature. The gardener activates heating if the temperature is too low.
3. **Precondition**: Garden is initialized.
4. **Postcondition**: Garden temperature is adjusted.

Simulate Pest Attack:

1. **Actor**: User

2. **Description**: The user simulates a pest attack on the plants. The gardener evaluates the impact based on plant susceptibility and pesticide application.
3. **Precondition**: Garden is initialized.
4. **Postcondition**: Plant statuses are updated based on the pest attack.

Apply Pesticide:

1. **Actor**: Gardener
2. **Description**: The gardener periodically applies pesticides to protect plants from pests.
3. **Precondition**: Garden is initialized.
4. **Postcondition**: Plants have enhanced resistance to pests.

Activate Sprinklers:

1. **Actor**: Gardener
2. **Description**: The gardener activates the sprinkler system when rainfall is insufficient.
3. **Precondition**: Rainfall is below the threshold.
4. **Postcondition**: Plants receive adequate water from sprinklers.

Activate Heating:

1. **Actor**: Gardener
2. **Description**: The gardener activates heating when the garden temperature falls below a safe level.
3. **Precondition**: Temperature is below the safe threshold.
4. **Postcondition**: Garden temperature is maintained at a safe level.

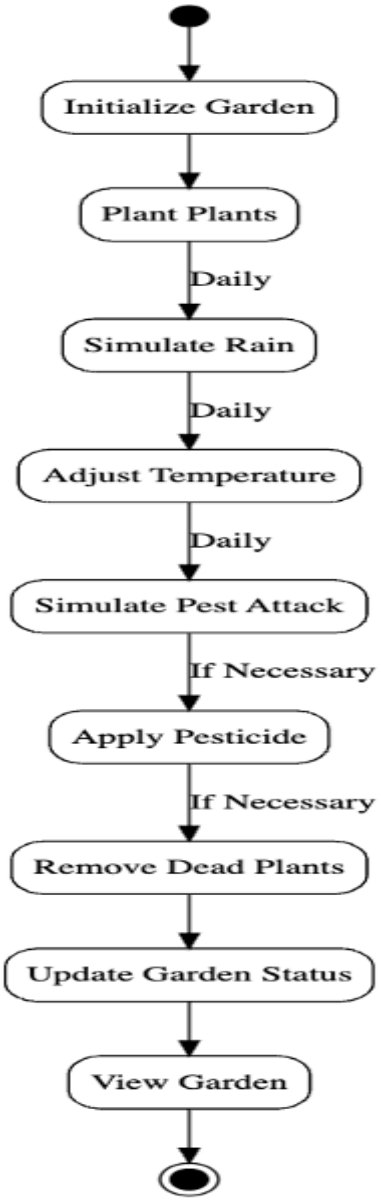
Update Plant Status:

1. **Actor**: Gardener
2. **Description**: The gardener updates the status of each plant based on the current conditions (water, temperature, pests).
3. **Precondition**: Garden is initialized.
4. **Postcondition**: Plant statuses are updated.

Log Events:

Actor: Gardener
Description: The system logs all significant events and actions for monitoring and debugging purposes.
Precondition: Garden is initialized.
Postcondition: Events are logged.

Activity Diagram:-



The activity diagram showcases the workflow of the garden simulation, including the following key activities:

Initialization:

1. The simulation is initialized through GardenSimulatorAPI.

2. Plants are loaded from a configuration file.

Daily Operations:

1. Each day, the simulation updates weather conditions such as rain and temperature.
2. Pest attacks are simulated, and pesticides may be applied based on probability.The status of each plant is checked and updated.

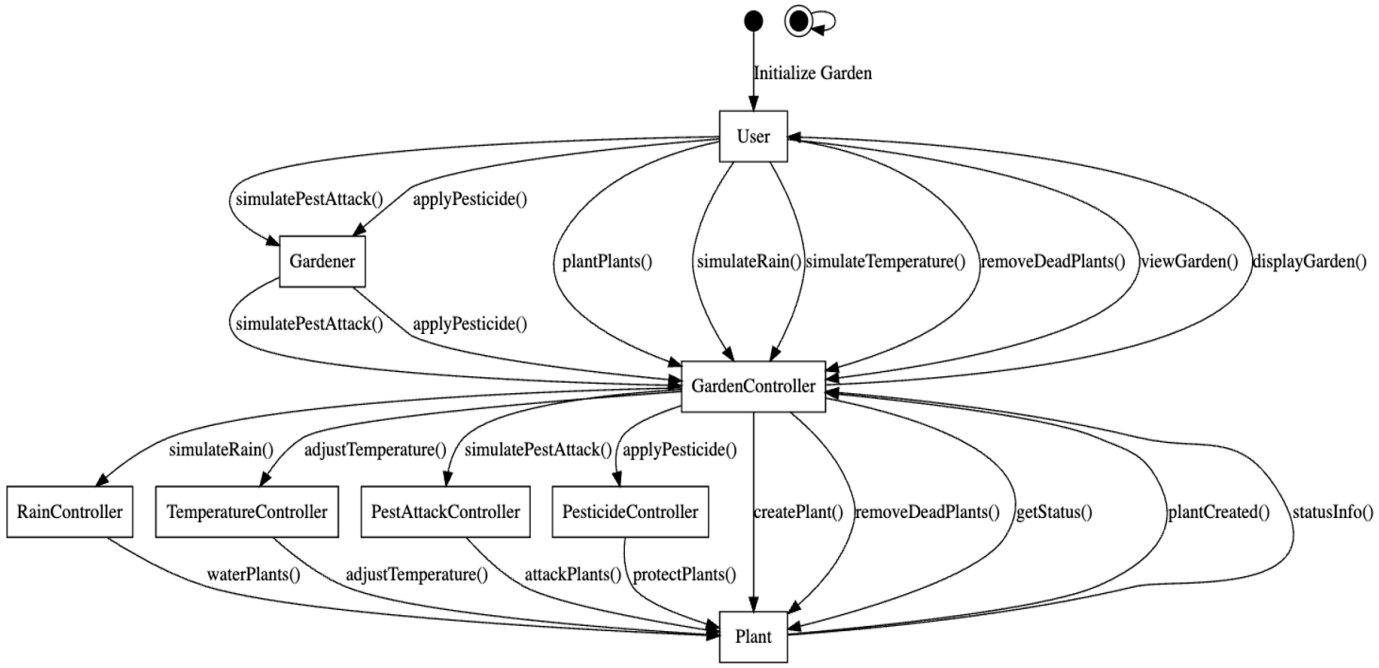
GUI Interaction:

1. Users can interact with the simulation through the GUI.
2. Actions like adjusting temperature and simulating rain can be triggered via the GUI.

Logging:

1. All significant events and actions are logged using Log4j.
2. Logs include rainfall, temperature changes, pest attacks, and plant status updates.

Sequence Diagram:-



Description:

The sequence diagram details the interaction between objects in the simulation over time, focusing on the flow of messages and calls. Here, we'll describe a scenario where a user interacts with the system through the GUI to simulate a day in the garden, including rain,

temperature adjustment, and pest control.

Sequence of Events:

User Initiates Simulation:

1. The user starts the simulation via the GUI (ViewController).
2. ViewController sends a startSimulation message to GUIMain.

Initialize Garden:

1. GUIMain calls initializeGarden on GardenSimulatorAPI.
2. GardenSimulatorAPI loads plant configurations from config.json and initializes the GardenController.

Simulate Rain

1. ViewController sends a rain command to GardenSimulatorAPI.
2. GardenSimulatorAPI invokes simulateRain on the RainController.
3. RainController checks if the rainfall amount meets the threshold; if not, it activates sprinklers via SprinklerController.

Adjust Temperature

1. ViewController sends a temperature command to GardenSimulatorAPI.
2. GardenSimulatorAPI invokes simulateTemperature on TemperatureController.
3. The TemperatureController adjusts the garden temperature and activates heating if needed via HeatingController.

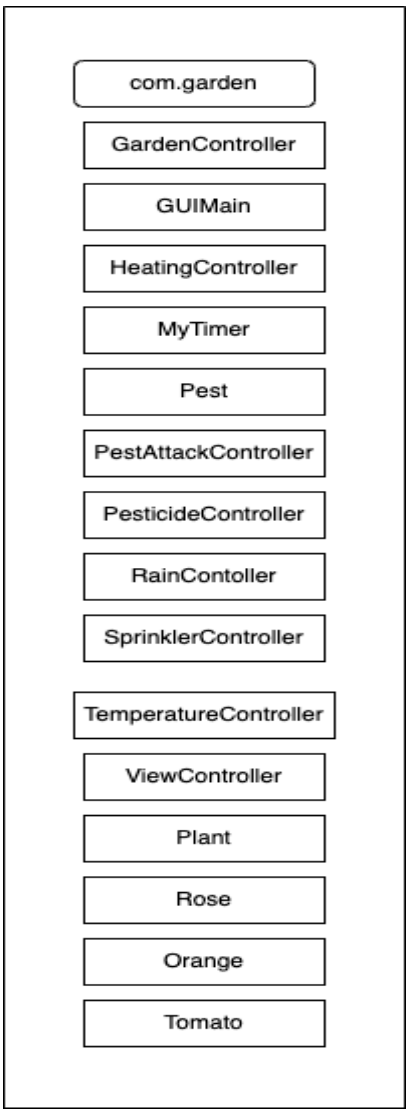
Simulate Pest Attack

1. ViewController sends a parasites command to GardenSimulatorAPI.
2. GardenSimulatorAPI invokes simulatePestAttack on PestAttackController.
3. PestAttackController iterates through plants and applies the pest attack logic.

End of Day

1. GardenSimulatorAPI calls sleepOneHour to simulate the passage of time.
2. The day counter is incremented, and the simulation logs the end of the day.

Package Diagram:-



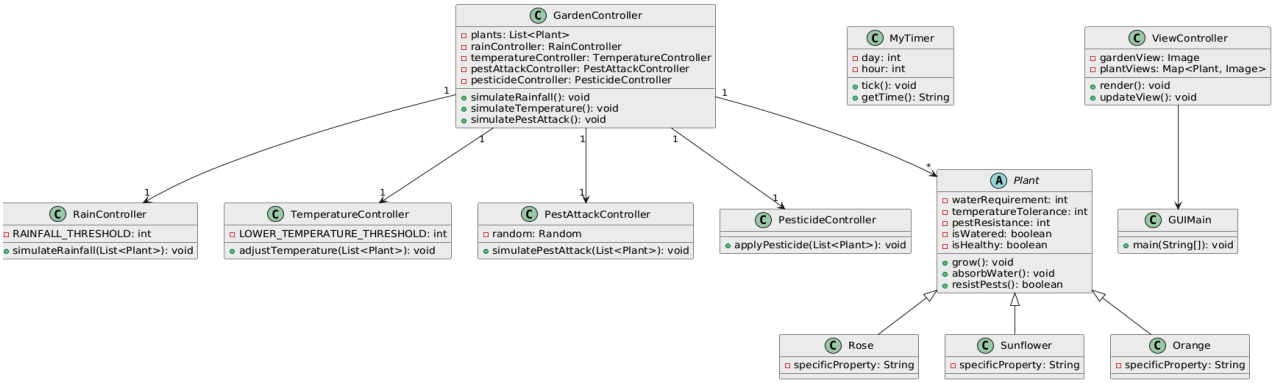
The package diagram outlines the structure of the project, showing the relationships between different packages:

1. **com.garden.controllers:** Contains all controller classes managing specific aspects of the garden.

2. **com.garden.models:** Contains model classes representing different plant types.
3. **com.garden.simulation:** Contains classes related to the simulation lifecycle and timing.
4. **com.garden.gui:** Contains classes related to the GUI implementation.

Class Diagram:-

The class diagram illustrates the relationships and interactions between classes. Key relationships include:



1. **GardenController** uses instances of RainController, TemperatureController, PestAttackController, and PesticideController to manage different garden operations.
2. **Plant** is an abstract class extended by Rose, Tomato, and Orange, each with specific attributes.
3. **GardenSimulator** and GardenSimulatorAPI interact to initialize and run the simulation.
4. **GUIMain** and ViewController work together to provide a graphical interface for the simulation.

Implementation :-

Core Functionality:

Initialization:

The simulation is initialized using the GardenSimulatorAPI class, which loads plant configurations from a JSON file and sets up the garden environment.

Rain Simulation:
The RainController class simulates rainfall. If the rainfall is below a certain threshold, the SprinklerController is activated to provide additional water to the plants.

Temperature Control:
The TemperatureController class adjusts the garden temperature. If the temperature drops below a minimum safe level, the HeatingController is activated to warm the environment.

Pest Management:
The PestAttackController class simulates pest attacks on plants. The PesticideController can apply pesticides to plants, enhancing their resistance against pests.

GUI Implementation:
The GUI is implemented using JavaFX. The GUIMain class loads the FXML layout and starts the application. The ViewController handles user interactions and updates the simulation accordingly.

Usage:
The Automated Garden Simulator provides a simulated environment for managing a garden through various environmental conditions and interactions. Here's how to use the system:

Starting the Simulation:

1. Launch the application using the main class GUIMain which initializes the GUI.
2. The GUI presents controls for various actions like simulating rain, adjusting temperature, and simulating pest attacks.

User Actions:

1. **Initialize Garden:** Use the GUI to initialize the garden. This action loads plant configurations from config.json and sets up the initial state of the garden.
2. **Simulate Rain**: Enter the amount of rainfall and submit. The system will water the plants, and if the rainfall amount is below a threshold, the sprinkler system will be activated.
3. **Adjust Temperature**: Set the desired temperature and submit. The system will adjust the garden's temperature, activating heating if the temperature falls below a safe level.
4. **Simulate Pest Attack**: Select a pest and submit. The system will simulate a pest attack on the garden, affecting plants based on their susceptibility to the pest.

Gardner Actions:

1. **Apply Pesticide:** Periodically, the gardener (an internal actor) will apply pesticides to enhance plant resistance to pests.
2. **Activate Sprinklers**: If rainfall is insufficient, the gardener will activate the sprinkler system to ensure plants receive adequate water.
3. **Activate Heating**: The gardener will activate heating to maintain a safe temperature for the plants if it drops too low.
4. **Update Plant Status**: The gardener continuously monitors and updates the status of plants based on water, temperature, and pest conditions.

Monitoring and Logging:

1. The system continuously logs significant events, including environmental changes and plant status updates.
2. These logs are essential for monitoring the garden's state and debugging purposes.

Logging:

The Automated Garden Simulator utilizes Apache Log4j for logging purposes. Logging is implemented across various components to capture key events and actions within the simulation. Here is an overview of the logging implementation:

1. **Initialization**:
    a. When the garden is initialized, the system logs the details of loaded plants.
    b. Example: log.info("Garden initialized with plants: {}", plants);

2. **Rain Simulation:**
    a. The system logs the amount of simulated rainfall and whether the sprinkler system was activated.
    b. Example: log.info("Simulating rain of {} units.", rainfallAmount);
    c. Example: log.warn("Insufficient rainfall received {} units which is below the {}. Activating sprinkler system.", rainfallAmount, RAINFALL_THRESHOLD);

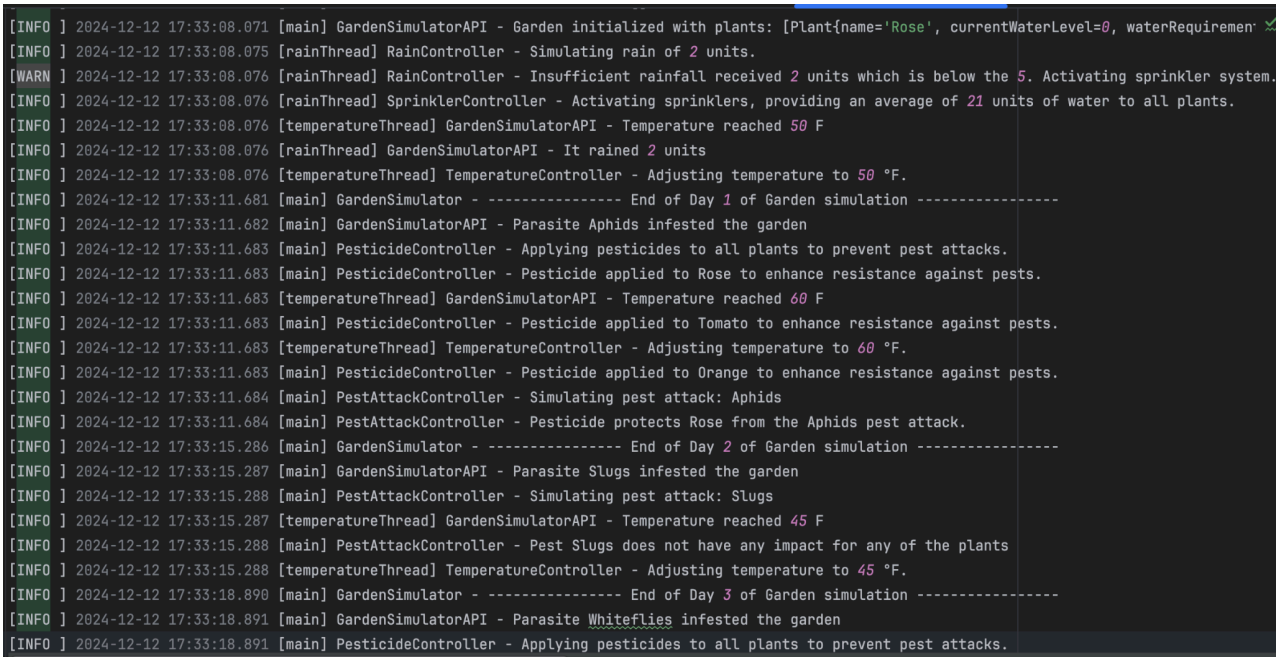
3. **Temperature Adjustment:**
    a. The system logs the new temperature set for the garden.
    b. Example: log.info("Temperature reached {} F", temperature);

4. **Pest Attack**:

a. The system logs the occurrence and impact of pest attacks on the plants.
b. Example: log.info("Simulating pest attack: {}", selectedPest);
c. Example: log.warn("Plant {} has been killed by a {} pest attack.", plant.getName(), selectedPest);

5. **Gardner Actions:**
   a. The gardener's actions, such as applying pesticides, activating sprinklers, and heating, are logged for monitoring.
   b. Example: log.info("Applying pesticides to all plants to prevent pest attacks.");
   c. Example: log.info("Activating sprinklers, providing an average of {} units of water to all plants.", averageWaterRequirement);
   d. Example: log.info("Activating heating system to increase temperature to {} °F.", MINIMUM_SAFE_TEMPERATURE);

6. **Plant Status Updates:**
   The system logs the status of plants, including whether they are alive or dead after environmental changes or pest attacks.
   Example: log.info("Alive Plants : {}", gardenController.getAlivePlants());
   Example: log.info("Dead Plants : {}", gardenController.getDeadPlants());



Conclusion:-

1. Summary of the Project:
    a. The Computerized Gardening System (CGS) is a comprehensive and automated solution designed to address the complexities of managing large and diverse gardens. By integrating advanced features such as automated watering, pest control, heating, and a robust logging system, the project showcases how modern technology can transform traditional gardening practices. Through a user-friendly JavaFX interface, the system simplifies garden management while ensuring sustainability and efficiency.

2. Key Takeaways:
    a. Automation for Efficiency: The system minimizes manual effort and human error by automating key gardening tasks.
    b. Modular Design: Independent modules ensure scalability and easier maintenance.
    c. User-Centric Approach: An intuitive GUI and detailed logs enhance usability for both technical and non-technical users.
    d. Resilience and Sustainability: The garden can operate autonomously over extended periods while optimizing resource usage.
3. Final Thoughts :
    a. Overall, the project demonstrates an effective blend of user interaction and automated management, providing valuable insights into garden care and management through a simulated environment. The use of UML diagrams, detailed logging, and comprehensive use cases ensure that the system is both user-friendly and robust, making it a useful tool for both educational and practical purposes in understanding garden ecosystems.

SANTA CLARA UNIVERSITY
SCHOOL OF ENGINEERING