**PROJECT DESIGN**

**<**Alchemilla – A Java-Based House-Plant Care Scheduling Desktop Application**>**

**Table of Contents**

[Project Abstract 3](#_Toc440365550)

[High Level Requirement 3](#_Toc440365551)

[Conceptual Design 3](#_Toc440365552)

[Background 4](#_Toc440365553)

[Required Resources 4](#_Toc440365554)

Product Vision 5

Personas 6

Feature List 8

Class Diagram 9

Sequence Diagrams 10

Weekly Progress 12

Testing 13

## Project Abstract

Alchemilla is a home plant-care scheduling solution designed to simplify the daunting but rewarding act of caring for plants at home. Aspiring growers will be able to upload information about house plants, or to use the program to find information about houseplants. They can then organize this information by selecting plants they wish to care for, and the program will generate a care schedule. This schedule can then be exported to a common calendar application such as Outlook Calendar. The name Alchemilla was chosen because this plant is sometimes referred to as the 'Forget-Me-Not'. The purpose of this application is to assist plant care enthusiasts in simplifying the management of their care routines, ensuring they never forget to water another plant. Upon completing a scheduling and adding it to their calendar, aspiring growers can rest assured they will have all of the information they need to competently care for their plants.

## High Level Requirements

At the user-level, the program will allow for importing houseplant information, and exporting that information to a calendar. Users can view the information in-app and can select which plants they wish to add to their care plan. The user can also store multiple plans and select between them. Each plan will be comprised of, at a minimum, a selection of plants owned or desired to be owned, and a start time for the plan. Each plant will associate with it, at a minimum, a name, picture (JPEG), watering time-deltas, feeding time-deltas as well as what food the plants need (if any), and a preferred lighting arrangement. Each plant will also have a difficulty rating associated with it, and the overall plan difficulty will be calculated as an average of the individual plants' difficulties.

The basic framework for the GUI will include a menu and button bar, a side panel from which to select plants currently stored by the program, and a display window to show the image of the plant and associated information. The side bar should also have an alternate frame that can be easily switched to which shows the plans the user has generated. There should also be a button bar associated with each side bar frame containing buttons for common actions such as starting a new plan or uploading a new plant.

## Conceptual Design

The current design plan for this program can be divided up into several component pieces. Each of these pieces could involve multiple classes.

1.) The Plant Interface: This component will be designed using the composition pattern. The intention is for the development team to begin by implementing user-uploaded plant information. In the future several other classes will implement this interface that automate the process of ingesting plant information. In all cases, this data will return to the Storage Module (see component 3).

2.) The Plan Generator: In the simplest terms, this is a class that maps Plant objects to a particular plan. The plan will have its own name, ID, and difficulty rating derived from the difficulty ratings of the individual plants. The plan generator should also have the ability to export the plans to a Calendar application such as Outlook.

3.) The Storage Module: This module will be responsible for managing the storage of data associated with Plant Objects (such as JPEG and care information) and care-plans saved by the user. Access to saved Plant objects and Plan objects will be made through this module, as will storage of new Plant objects and new Plans. The current vision for storage is to assign each plant a unique ID, which is used to access data about the plant from a columnar data source. JPEGs will be saved with the ID as their name. Plans will consist of a list of Plant IDs, as well as necessary meta-data such as plant start date and the calculated difficulty of the plan.

4.) The GUI manager: This component will communicate directly with the Storage Module. The GUI manager will give commands to the Storage Module to set Plan information. Information returned from this module, such as plans or individual plants, will be used to populate GUI frames. The GUI manager will need to implement a variety of Event Handlers for buttons and menu actions. Some of these actions will change the Plan meta-data, such as the name of a plan, the plants associated with the plan, or the start date of the plan. GUI frame design will be implemented using the GUI design tools included with IntelliJ IDEA IDE.

## Proof of Concept

<https://github.com/Alexander-Russakoff/alchemilla>

## Background

The main draw of this application over others will be the calendar exporting feature. Other similar applications offering information on caring for houseplants orient themselves toward user engagement via active logging. Users are often expected to navigate the resources themselves, and to then record their plant care activities and the progress of their plants with-in the app. Often these apps are based on an economy of in-app purchases or serving ads. The incentive structures at play result in applications that demand time and energy of the user. This application, developed as an open-source project, hopes to deliver a solution that gives time and energy back to the user, providing value through simplifying rather than ‘gamifying’ the craft of house-plant care.

Chu, Haidee. “7 Apps to Keep Your Plants Alive and Well.” Mashable, Mashable, 29 Oct. 2021, https://mashable.com/article/best-apps-for-taking-care-of-plants.

## Required Resources

Building this software will require developing knowledge of several resources. Though some have already been mentioned above, this section touches on some of the key requirements more explicitly.

All code will be written in the Java Programming language. The development team will need basic familiarity with Java (up to v14) and Object Oriented Programming.

All code will be developed in the Linux operating system. There are several reasons to begin development in Linux, and conform to POSIX standards. The hope is that this will be the simplest starting point for moving the program to other OS ecosystems in the future. Currently, Mac OS conforms to POSIX standards, and Windows is planning to implement the Linux kernel in a soon to be released version of Windows 11. Additionally, many courses at Temple University require the use of a Linux environment, so other students who join the development team are likely to already be familiar with the OS.

A key piece of functionality will be the ability to export to Outlook, and eventually Gmail. At a minimum, developers will need to learn how to use the Microsoft Graphs 1.0 API to generate and export a calendar schedule. This will likely also involve learning the Java Calendar library.

Developers will need to develop a basic understanding of Java GUI construction. The most likely candidate for GUI development is the Swing Framework, however another framework might be chosen. At a minimum, this project will require implementing multiple frames, implementing event handling for buttons, drawing JPEGs to the screen, and several other basic GUI features. At this time, IntelliJ’s integrated GUI design tools are the preferred method of designing frames.

Gathering and storing plant information will be implemented using the composition pattern, with several classes inheriting from a Plant Interface. The reason for this is the program is intended to be able to generate plant information in multiple ways. The design roadmap will begin with users uploading information based on a template. Next, an API will be leveraged to attempt to generate that information automatically. Finally, and time permitting, a 'web-scraper' will be implemented that can gather plant information from the internet.

The development team will need to become familiar with several open-source API that aggregate data about houseplant and house-plant care. The following are some candidate data sources:

https://plantdatabase.kpu.ca/

https://www.producthunt.com/posts/trefle

https://www.growstuff.org/policy/api

If initial design goals are reached, the development team will need to familiarize themselves with 'web-scraping' basics. The current candidate packages for this phase of development are JSOUP and Selenium.

## Product Vision

For busy people who need an automated solution to plant care, small and large scale, Alchemilla is an application that provides detailed plant care schedules, social sharing functionality, and calendar exporting features that assist and enable users in plant care. Unlike other services, our product offers calendar integration with popular calendar applications, like Outlook, as well as business and multi-user/social sharing capabilities.

## Personas

*Jacob, a plant-care beginner*

Jacob, age 38, is a member of the millennial generation who remembers the old days of the internet when personal data harvesting for the purposes of advertising was very rare. He is well educated, having a master's degree in architecture, and works for a mid-size firm designing office parks. He has recently had his first child and worries about their future social media use. He strives to make his home more comfortable and inviting for his family. Jacob is interested in learning how to care for houseplants and has a basic idea of what it will entail, but he is concerned his work and his family will cause him to struggle with maintaining them. That the application pushes the schedule to his personal calendar works well for Jacob as he would prefer to spend as little time setting up and maintaining the schedule as possible. He is also aligned with the designer's ideal of not collecting user information.

Author: Alexander

*Randal, a plant-care expert*

Randal, age 32 is an engineering technician with an associate degree in mechanical engineering. He works in industrial manufacturing as part of quality and operations testing, and is an avid gardener, caring for his large host of house and outdoor plants in his free time. Randal is quite well-versed in his hobby, researching and caring for his plants on his own time, but has recently come upon a busier work schedule and overtime work, eating into the amount of time he can expend with plant care. He has tried searching for tools that would simplify his plant work but only encountered apps that were either overcomplicated with too many functions he did not need or apps only providing an initial free trial or limited functionality with a subscription needed thereafter. Randal is familiar with setting up a care schedule but manually doing so for all his plants would be too time-consuming and he would like a program with just the simple functionality of generating a calendar without all the accessory functions he wont use or the additional care information he already knows.

Author: Akshay

*Maria, a small business owner*

Maria, aged 53, is a small business owner with a bachelor's degree in accounting. Her business is a small storefront in a mid-sized city where she sells locally sourced hand-crafted goods. She spends a lot of time in her store and would like to liven it up with a selection of easy to care for plants. If this goes well, she thinks, then maybe she will begin to sell potted plants in her store as she feels it would match the aesthetic of her other wares. Maria is not particularly tech savvy, but she is adept with email and basic accounting software she uses in the course of her business. She would like some assistance in caring for her plants, reminders for when to water and feed the plants and so on. Having the schedule hook into her work calendar is ideal for her.

Author: Alexander

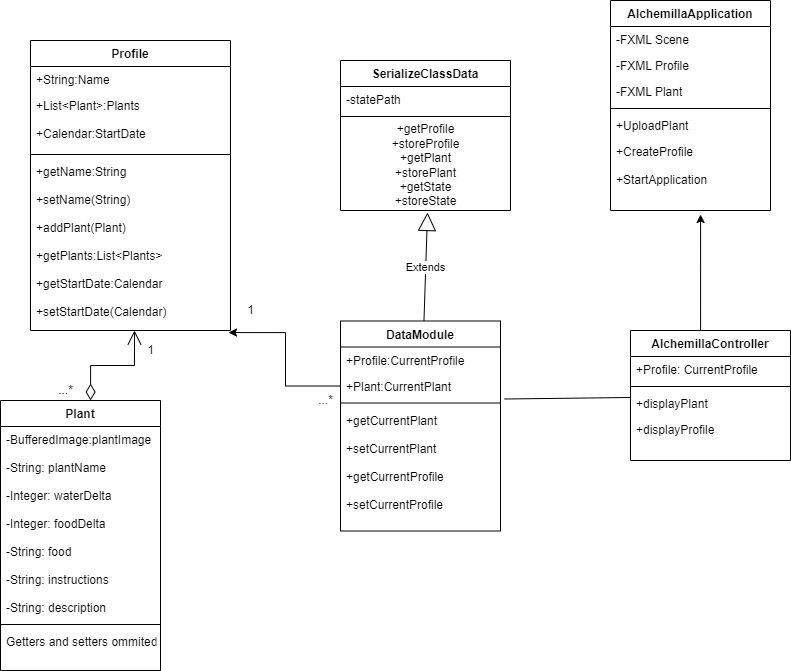
*Winona, a community garden leader*

Winona, age 37, is a busy marketing director living in the Fairmount area of Philadelphia. She is highly educated, getting a bachelor’s and an MBA in marketing. She is adept with technology and is quick to learn as the ever-shifting marketing landscape demands it. She lives a busy life, balancing her chaotic work life with being a mother to two children. But with all of that, Winona still maintains an active and healthy social life and loves the community around her. When she’s attending community events, working, or going to her children’s various extracurricular activities, Winona loves to set aside time for her true passion: gardening. A regular at the various botanical gardens in and around the Philadelphia area, she would love to have the opportunity to do some gardening herself but does not have the time to tend to a garden herself. Thankfully, a new community garden opens up - the perfect opportunity for Winona. A natural born leader, she is interested in Alchemilla’s organizing capabilities. She would like to use Alchemilla to generate schedules that are easy-to-integrate into her own daily life, which revolves around her meticulously planned Outlook Calendar. Similarly, Winona is interested in Alchemilla’s ability to add people to plant-care schedules, enabling a more organized approach to the shared garden space.

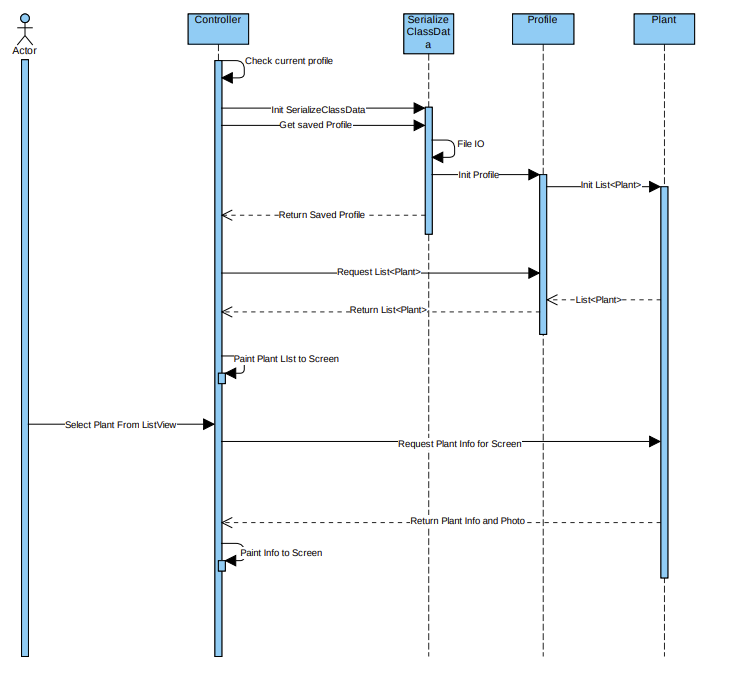
Author: Jason

## Feature List

* Amalgamate and Display Information
  + For invested plant care enthusiasts who consider purchases and learning to cultivate new plants as similar to a collection game. This user enjoys watching their collection grow, both literally and in a consolidated space via the application. Alchemilla is a simple interface where they can store information they’ve collected about their plants and take pictures and upload them to each plant profile. Since the parameters such as description, watering times, and so on can be uploaded and modified by the user, this feature can be thought of as a personal plant care journal.
* Generate Schedule
  + For new plant care hobbyists or those curious as to how to begin caring for houseplants. This user knows they are not accustomed to the art of plant care and seeks guidance in performing the tasks necessary to keep them healthy. Alchemilla helps this user to gather the information they need, and extrapolate that information out to a long term, easy to follow schedule they can use to jump start their home plant care ambitions.
* Export Schedule to Calendar
  + For the busy working professional who is not particularly tech savvy, this feature allows them to merge their schedule with a technology they are already familiar with. They’re engagement with the application will be very low long term. This user wants to know they can set up a schedule once, then forget about it.
* Shared Calendars Between Users (Long Term Stretch Goal?)
  + We hope to implement a shared calendar feature, aimed towards plant-care routines that involve more than one person, such as a community garden. This feature would allow collaborative plant-care schedules for groups of people, as Alchemilla would divide the plant-care in an equitable fashion between users. This would require an account feature to be able to assign users to schedules. This feature could be a potential revenue stream as a “premium member,” giving users the option of extended functionality beyond the baseline personal plant-care functionality.
* Tracking Business Metrics (Long Term Stretch Goal)
  + In the future, we hope to expand Alchemilla to contain a business logic and metric tracking component aimed toward small and medium sized businesses. This feature is a revenue stream that helps ensure the goal of keeping the application 100% free of data harvesting practices and predatory advertising while also keeping the core features free for the average non-commercial user.



Program Class Diagram This class diagram describes the structure of the program and informs its functionality. The primary data storage class for plant information is the Plant class which includes fields detailing all the key information necessary about the plant and its care. The objects created from this class are stored in association with a Profile, which describes a specific user and has an associated list of plants as well as a calendar object; it also functions for altering profile information, adding plants to the profile or changing schedule information. The profile then feeds into the application’s overlay, which involves the DataModule that stores both the current Profile and current Plant to be displayed as well as methods to obtain other plants and profiles; this class extends the SerializeClassData which handles obtaining and storing profile and plant information. In terms of the display, AlchemillaController handles the actual display of the application with a field for the current profile and methods physically displaying profile/plant information. Finally, AlchemillaApplication serves as the physical layout for the application, holding layout information for the display scene, profile and plant; functions within this class allow for adding new information including uploading plants, creating profiles and starting/updating the application.

****

Plant Addition Sequence Diagram This diagram describes the addition of new Plants to a user Profile and then their updated display on the Profile’s list. As with the standard viewer function, the first operations are completed by the Controller which checks the current user Profile and then initializes SerializeClassData to obtain a saved Profile. The SerializeClassData then accesses files to obtain the associated Plant objects for the Profile specified. The Profile is then initialized and the associated List of Plant objects created to return to the Controller to display to the app screen. When a Plant is then selected from the ListView, the Controller requests information from the Plant object and then displays it to the user through the Plant description, information, and picture panes.

Chart, box and whisker chart

Description automatically generated

Plant Addition Sequence Diagram This diagram describes the addition of new Plants to a user Profile and then their updated display on the Profile’s list. As with the standard viewer function, the first operations are completed by the Controller which checks the current user Profile and then initializes SerializeClassData to obtain a saved Profile. The SerializeClassData then accesses files to obtain the associated Plant objects for the Profile specified. The Profile is then initialized and the associated List of Plant objects created to return to the Controller to display to the app screen. When the Plant addition option is selected, the associated window is utilized to input the new plant information. This new Plant object is then returned to the Controller and which passes it to the SerializeClassData to save the Plant at the specified Profile file location. The Controller then calls the Profile again to retrieve the new Plant List (with the new Plant added) and return it to the Controller to display the updated list on-screen.

## Week 2 Progress

**Sprint Goal:** The goal for this sprint is to create a main scene and GUI template for plants and profile information in addition to creation of a class diagram.

**Backlog Features**

* Develop GUI template for storing plant information
* Develop GUI template for storing profile information
* Create main scene template for application

|  |  |  |
| --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To |
| Plant Information Template | Partially Completed | Alex/Jason/Akshay |
| Profile Information Template | Partially Completed | Alex/Jason/Akshay |
| Class Diagram | Completed | Alex |

## Week 3 Progress

**Sprint Goal:** The goal for this sprint is to create a functional dynamic scene able to store and display plant information; in addition, creation of functional sequence diagrams and exploration of alternative tools for layout.

**Backlog Features**

* Dynamically update main scenes
* Pass ClassData and Profile between scenes
* Resolve file storage
* Create JSON parser for storing information
* Explore TornadoFX

|  |  |  |
| --- | --- | --- |
| Tasks in Sprint | Task Status at end of Sprint | Assigned To |
| Dynamically updating scenes | Partially Completed | Alex/Jason |
| Create JSON parser | Partially Completed | Alex |
| Pass Class Data/Profile | Completed | Jason |
| Resolve file storage | Completed | Alex |
| Explore TornadoFX | Completed | Akshay |
| Sequence Diagrams | Completed | Alex/Akshay |

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Testing for the program was done utilizing two different means due to the inclusion of GUI elements in addition to standard classes with data and functionality. Testing was completed for classes like Plant and Profile (which are primarily data-storage classes with methods to modify data) utilizing JUnit testing. This tested much of the basic functionality of the classes including the inclusion of relevant data fields in both as well as the “get” and “set” functionality of the data values in both. In addition to this, basic profile information testing was also done via JUnit for the plant controller.

For other testing, a class called Scrap was utilized to successfully test the GUI and proper functionality. Testing within this class involved including various numbers of plants, dynamically repopulating plant objects in the layout, displaying images based on plant URL, altering plant orders in display, manipulation of the scene displaying plant data, and creation/alteration of multiple instances of the program.