Thales Protopapas
5/29/2025
Holmer CSA Period 6

Final Project Written Report: Plantabase

The problem I set out to solve with my software artifact was the issue of keeping track of a large set of green walls within a school building. Green walls, or living walls, are arrays of plants set up indoors which help reduce students' stress and filter pollutants out of the air. Studies have demonstrated these benefits, and Brooklyn Technical High Schools' environmental science major, led by Dr. Margarita, has set them up throughout the building. However, keeping track of so many plants in a building as notoriously large as Brooklyn Tech's can be difficult. The plants must all be watered routinely, and the intervals for this watering and amount of water needed can vary between plant types.

The program I created allows users to input data about green walls within a school and the plants they contain. Walls can be defined and populated with plants of different types. A wall can be searched for an example of a certain type of plant. A formatted list in alphabetical order of every type of plant in the school and their total quantities can be printed. Days can be incremented as they pass and each plant will keep track of how long since it has last been watered. A formatted output describing which plants currently need to be watered and how much water is needed, both for each and in total, can be printed. Finally, once these instructions have been followed, the program can reset the water state of all plants which had to be watered that day.

The project has three main classes, Plant, which represents a single plant, Wall, which represents a green wall of plants, and School, which represents an entire building worth of walls. Plant has two instance variables, a String representing its type, and an int representing its "waterAge", the number of days since it was last watered. It has a constructor which defines its type and sets its waterAge to zero, accessor methods for both variables, a method to increase its waterAge by one day, a method to reset its waterAge to zero when it is watered, a toString method which returns a formatted output of its instance variables, and accessors for the plant type's watering interval and amount needed.

In order to define how much water different types of plants need and how often they need to be watered, I decided to use an external file in the project directory which can be imported from elsewhere and edited outside of the program. The file type is csv, comma separated values, and contains a list of plant names and their corresponding water intervals in integer number of days and water amounts in decimal number of liters. I had to learn about several new methods online to add file functionality to the program. This file is read within the Plant class when getWaterInterval and getWaterAmount are called. These methods iterate through the dictionary csv to find the row with the Plant object's type using the read method, which reads a given row

and column of the csv's data.

The Wall class also has two instance variables, a String name to identify it and a 2d array of Plant objects representing its physical grid of plants, satisfying one of the data structure requirements. Its constructor initializes its name and the 2d array with a given height and width. The populate method checks for empty locations within the 2d array and returns false if any of them are full. If the full range of rows and columns given is free, the method returns true, constructs Plant objects with a given name, and places them in the given range of the 2d array. The findPlant method takes a String for a plant type and linearly searches the 2d array for an example of that plant, returning an int array of length 2 representing its coordinate pair within the wall's grid and meeting the searching requirement. It has a method to increase the waterAge of every plant within it, accessor methods for its height, width, name, and a Plant within it at a given location, and a formatted toString method.

It has two methods used to deal with plants which must be watered. toWater returns a formatted String describing which plants at which locations within the wall must be watered. It iterates through the 2d array, checking each Plant for a waterAge greater than or equal to its water interval. It includes the amount of water needed by each plant as well as the total across the entire wall so that students traveling to water the wall know how much to bring. The second, needWater, operates similarly but returns an ArrayList of int arrays, satisfying the other data structure requirement and operating in a way useful to methods in the School class.

The School class represents a whole building and has a single instance variable, an ArrayList of Walls, initialized in the class's constructor. It has a method to add a Wall that has already been constructed to the School's ArrayList and one to increase the waterAge of every plant within every wall in the school. It has a toWater method which formats the combined outputs of each Wall's toWater method for a full set of watering instructions and a waterAll method which uses the needWater method of each Wall to update every plant which has just been watered.

Finally, it has a method to return a formatted list of every type of plant in the school and their total quantities. This method first iterates through every Wall in the School and creates an ArrayList of every Plant. Then it iterates through this and creates another ArrayList of every plant *type*, by checking the current type ArrayList for each element of the Plant ArrayList and only adding it if it does not already exist there. It then iterates through the type ArrayList and creates a new ArrayList with the same number of elements, but this time each String has both the type and number of plants of that type, calculated by iterating through the Plant ArrayList. Finally, a selection sort helper method, which I modified to use compareTo to order the ArrayList alphabetically, is called, satisfying the sorting requirement, and the finished String is returned for printing.

While discussing the project with my friends from the environmental science major, I learned that certain walls are already set up to automatically water themselves. To allow these to

also be organized in my program, I created a subclass of Wall called AutoWall. This meets the inheritence requirement as well as the polymorphism requirement, since AutoWall object references are stored in the Wall ArrayList within the School class. An AutoWall behaves the same as a normal wall, except its incDay method keeps every Plant's waterAge at zero to avoid displaying any plants as needing to be watered, and its toWater and toString methods specify that it is an automatic wall. Since the plantArray and name instance variables of the Wall class are private, AutoWall uses the accessor methods from the Wall class, as well as the super keyword, to achieve its functionality.

My biggest challenge outside of the csv dictionary was keeping track of the purposes of each method within each class. I would come back to the project after taking a break for a few days and totally forget what had to be called from the Wall class or the School class and which method gave a formatted output or a data structure. After getting my bearings back on my code, I left comments at the top of each method that whose purpose was not immediately obvious, detailing its use case, return value, and impact on the data within its class.

To use the program, the dictionary of plant information csv should be created and formatted like the example given in the source code directory. Walls should be constructed and populated with Plants as they exist in real life, defined as AutoWalls when applicable. Then the School should be constructed and have all Walls added. Each day, the School's incDay method should be called, then its toWater method should be printed and have its instructions followed. Once this is complete, the waterAll method should be called to update the database. An example of this routine exists in the Main class of the provided source code. Methods to print Walls, search for Plants within them, or display the School's population of Plants can be called at any time.