CSC 210
PA5 Garden
Spring 2025

PA5-Garden: this is a two-part assignment.

Part I
Submission: Submit a pdf of the UML diagram of your project to Gradescope.

Part II
Submission: Submit the full implementation of your Java files for the assignment to Gradescope. All of your Java files must include the following package name:

```
package com.gradescope.garden;
```

Learning Objectives

The goal of this assignment is to design a class hierarchy to solve a problem, implement that hierarchy, use polymorphism, and use two-dimensional arrays.
The class diagram you will be submitting will be a simplified version of the UML class diagrams. There are many resources on the internet for UML class diagrams, however, we have discussed the format expected in lecture. Refer to the class slides for examples.
Use the following notation:

- `-` for `private`
- `+` for `public`
- `#` for `protected`

Remember to include in your diagram:

- inheritance relationships with lines and arrows
- class name at the top of each box
- fields with types
- methods with parameter and return types
- access modifiers (-, +, #)

Garden Simulation

The problem you will be solving is a garden simulation. The simulation will read commands such as PLANT, PRINT, GROW, and HARVEST from a file and execute those commands. The garden that you will be implementing will consist of a number of rows and columns of plots. Within each plot there can exist a single plant, which is represented with a 5x5 grid of cells. Plants are divided up into three different categories: trees, flowers, and vegetables, all of which have unique characteristics. For example, trees grow up, vegetables grow down, and flowers bloom as they grow.

Assignment

This is a **two-part assignment**. The class inheritance diagram is the first part, and the Java implementation is the second part.

- Class inheritance diagram: Create this diagram however you would like (draw it by hand, use software). However you create your diagram, you ultimately need to create a pdf file of your drawing and submit it to Gradescope.
- Your main program, which must be named **RunGarden.java**, must take the name of an input file on the command line, just as the main program for the Spotify assignment used a command line argument for the song file.

Sample Input/Output: The input file will contain the garden initialization settings and the commands to simulate the growth of the garden. Here is an example input file:

```
rows: 1
cols: 1
PLANT (0,0) banana
PRINT
GROW 1
print
```

Note that the commands should be case-insensitive. In other words, “print”, “PRINT”, and “Print” are all equivalent in the input file. Here is the output for the example:

```
> PRINT

.....

.....

.....

.....

..b..

> GROW 1

> PRINT

.....

.....

.....

..b..

..b..
```

The output should be printed to standard out. See the **PublicTestCases.zip** on the class website for more input and output examples. Note that in the above example, we asked for 1 row and 1 column. That gives us one plot at (0,0). We then plant a banana in the one plot at (0,0). Since a banana is a tree (more on that later), it starts in the bottom middle of the plot.

The following are the types of specific plants that could be planted:

FLOWERS	TREES	VEGETABLES
Iris	Oak	Garlic
Lily	Willow	Zucchini
Rose	Banana	Tomato
Daisy	Coconut	Yam
Tulip	Pine	Lettuce
Sunflower		

Plants will be represented with ascii characters.

- The lowercase version of the first letter of the plant name is used to represent the plant. For “Garlic” the letter ‘g’ is used, for “Daisy” the letter ‘d’ is used and so on.

Each plot in the garden contains a 5x5 grid of “cells”, represented as characters, that hold the plant in that plot. (Each location in a plot is called a cell.) Here are the rules for initial plant placement in a 5x5 plot grid:

- Flowers start in the middle
- Vegetables start at the top middle
- Trees start at the bottom middle

Below are examples of plots for different types of plants:

```
Rose
.....
.....
..r..
.....
.....

Tomato
..t..
.....
.....
.....
.....

Coconut
.....
.....
.....
.....
..c..
```

Commands

This section lists the commands that must be implemented and provides an example for each command. See the **PublicTestCases.zip** on the class website for more examples.

- PLANT
  Example: PLANT (0,0) rose
  If the PLANT command is read, it should be followed by plot coordinates and the type of Plant to be planted. Use this type to plant the correct subclass of plant into the garden at given plot coordinates. The plot coordinates are given as row and column. Both rows and columns start at 0. Rows go down the screen, and columns go across the screen. Each plot will itself contain a grid of 5x5 cells (represented as characters). There is a restriction that the number of cells across should be less than or equal to 80, therefore the most plot columns allowed is 80/5 or 16.

- PRINT
  Example: PRINT
  If the PRINT command is read, then the entire garden should be printed to standard out.
- GROW
  Example: GROW 1
  If the GROW command is read, then each Plant should grow the specified number of times as seen in the input command. A plant cannot grow out of its plot. No error will happen, but growth should not occur outside the plot boundaries. Plots also cannot run into each other.

- GROW [num] (row,col)
  Example: GROW 1 (2,3)
  Grow whichever Plant is located in the garden at position (row,col) num times. If there is nothing at this position or the position is outside the size of the garden, print, “Can’t grow there.” and continue.
- GROW [num] [type]
  Example: GROW 1 rose
  Grow only Plants of the specified type num times.
- GROW [num] [plant]
  Example: GROW 1 flower
  Grow only Plants of the specified class num times.
- HARVEST
  Example: HARVEST
  Remove all Vegetables from the Garden.
- HARVEST (row,col)
  Example: HARVEST (2,3)
  Harvest Vegetable at location (row,col). If not a Vegetable or outside of Garden, print, “Can’t harvest there.” and continue.
- HARVEST [type]
  Example: HARVEST tomato
  Harvests all Vegetables of the specified type. If there are no Vegetables with that type, do nothing.
- PICK
  Example: PICK
  Remove all Flowers from the Garden.
- PICK (row,col)
  Example: PICK (2,3)
  Pick Flower at location (row,col). If not a Flower or outside of Garden, print, “Can’t pick there.” and continue.
- PICK [type]
  Example: PICK rose
  Pick all Flowers of the specified type. If there are no Flowers with that type, do nothing.
- CUT
  Example: CUT
  Remove all Trees from the Garden.

- CUT (row,col)
  Example: CUT (2,3)
  Cut Tree at location (row,col). If not a Tree or outside of Garden, print, “Can’t cut there.” and continue.
- CUT [type]
  Example: CUT PINE
  Cut all Trees of the specified type. If there are no Trees of that type, do nothing.

Error Handling

- Some of the commands above specify some error handling. Be sure to handle the errors specified.
- The garden should never be more than 80 characters across. If it is, your program should print out the message “Too many plot columns.” and then end. Think: How many characters across is each plot?
- Other than the above specified errors, the input can be assumed well-formed.

Design Suggestions

We recommend the following:

- A Garden object with a 2D array or list of plant objects.
- A Plant class hierarchy of some kind.
- A Screen object with a 2D array of characters that each plant can “print” its current representation into. The Screen object can then print everything.

  Hint: when copying over a Plant’s representation array, place each cell at

  [(Plant's row * 5) + cell row][(Plant's col * size of plot) + cell col]

  into the Screen object’s Array.

This assignment can be a lot more difficult or a lot easier depending on how well thought-out your solution is. I would recommend spending quite a bit of time—before programming—thinking about which classes you should have and the inheritance relationships between them.

Grading Criteria

For this PA, we are providing the test cases in **PublicTestCases.zip**, but we will also be testing your code on other testcases. Make sure you do the same.

60% of this assignment grade will be correctness.

The other 40% of your grade will be your decomposition and code clarity. Twenty of these points will be on your class hierarchy diagram.

Hardcoding: You will be deducted for “hardcoding”, that is, writing methods or code to make specific test cases work. You should not hardcode the cases for growing a flower.

Decomposition:

- Points will be taken off for copy, pasted, and edited code that should have been encapsulated in a method.
- **This program should use fewer than 10 .java files.** Each of these files should be (<275 lines).
- Each static method should be less than 30 lines. This INCLUDES comments, but not the method header. It is easier to read a function if it can all fit on one screen.
- Make things as simple as possible.
    - Only use one Scanner instance.
    - Don’t use lambda functions or other features in non-standard ways.
    - Avoid nested loops unless they would be cleaner when handling a 2D array.
    - Avoid nesting conditionals.

Testing:
The autograder reports on whether you passed or failed a test case, but if you fail a test case, it does not give you information on why your code failed. To help provide information on a failed test case, I am providing a JUnit test case that you can use (and modify) to test your code.

On the class website under the Assignments link, there is a file called **RunGardenTest.java**. This contains a JUnit test case specifically for the input file `plantcool.in` and the corresponding expected output file `pa5-plantcool.out`. Run this JUnit test case to test the `plantcool` test.

You can add additional test cases by copying the JUnit test case for `testOne()` and changing the `.in` and `.out` file names for new test cases.

Academic Integrity:

Write your own code. We will be using a tool that finds overly similar code. Do not look at other students’ code. Do not let other students look at your code or talk in detail about how to solve this programming project. Do not use online resources that have solved the same or similar problems. It is okay to look up, ”How do I do X in Java”, where X is indexing into an array, splitting a string, or something like that. It is **not** okay to look up, ”How do I solve this programming assignment from CSc210” and copy someone else’s hard work in coming up with a working algorithm.