Manuel Badena, Jayden Kang, & Ellie Yang

Desired Grade: A+

Functionality

First and foremost, our program runs (yay). It responds to inputs as described, and will not glitch if you go crazy with the keyboard or the mouse. Our user interactions are mostly through JButton, so there aren't any other errors that would occur based on user input. The game will also run forever and what the user sees each round in the garden is based on their actions and inputs.

Design

Our design was clear and concise from the beginning. Honestly, did you even play the game? Clearly beautiful. We have a cohesive sizing and color scheme. We created a loading screen and start screen so that the user can be introduced to the game and its basic concept before they jump into the game. This also ensures that they don't get thrown right into the night/day cycle animation. We also coded the windows so that each screen followed another (home night/day cycle → sorting screen → score summary → garden → night/day again). This creates a streamlined experience for the user and each next step is made clear with the labeled buttons and the writings on the screen. In particular, the way the score summary relates to the user's sorting by displaying the items they sorted and their correctness interacts well and creates a satisfying experience for the user. Their score then also relates to the flowers' growth in the garden (which are animated to add life), once again adding to the cohesiveness of our overall design. Overall, we would say that the way each stage of the game influences the outcome of the next is the best part of our design.

Creativity

This is one of the strongest parts of our project. Our program is completely original. We haven't seen anything else like it on the internet, and everything from visuals to game logic was created by us (we even have a logo). We think our program really encompasses our personal characters and merges it with an educational composting game.

Sophistication

Our program definitely took way more effort than any other lab or project we have done. It is also much more sophisticated as we included many Java libraries and multiple classes that all interact with each other in countless ways to form a cohesive game. Some libraries that we

implemented would be Java IO file input, Java Util (Scanner and ArrayList), Java Sound Audio Systems, etc.

Our program also includes several moving pieces that we have written to flow together smoothly for a better user experience. With several instances of each class running in the background, each holding its own thread, there is a clear concern about exception errors and crashing systems. Our group, however, fit every piece together with a refinement that allows it to run forever without any runtime issues. All the pieces not only work off each other's fields and logic, but the user input also influences the appearance of each screen, meaning we needed to incorporate several methods and safety nets to account for potential user-caused errors.

Broadness

We used the following six elements: To obtain full credit, you should use at least 6 of the following in some justified way.

1. Java libraries that we haven't seen in class
    a. Java IO file input
    b. Java Swing (JButton; buttons for many aspects of the game)
    c. Sound Audio Systems (Background Music)
    d. Math (Math.random)
2. Subclassing (extends a class you created)
    a. Item Class – extended by every compostable object's class (40 total)
        i. Without this, the 40 individual items classes would be impossible to store as an array, which we do later for many parts of the sorting process
3. Interfaces (implements an interface you created)
    a. Goods – implemented by every compostable object's class (40 total), contains all the methods we needed
        i. This helps us to set up our item classes' structures which is essential to our code's organization
4. Built in data structures
    a. ArrayList, Array
5. File input/output
    a. FunFact Class takes an file, "facts.txt," and takes a random sentence for the facts
        i. This makes the fun facts logistics much less complicated and easy to edit
6. Randomization
    a. Our item list is random and fact list is also random
        i. Without this randomization, our game would be simple and the user can easily predict the next iteration

Code quality

Code Quality:

We have comments throughout the quote so that both us and others can understand the code whenever they go through it. All of our classes and variables are named based on their functionality. Our code is organized and easy to parse, which was especially important because we had so many item classes that it would be impossible to understand without organization and good naming. We went back and cleaned up our code so that it is concise, taking out any redundant/unnecessary lines that built up in the process.

Product Quality:

We believe that our final product looks professional. We (taking from Prof Alfeld's expertise) added a loading screen and start button, which made it look much more complete than just jumping right in whenever it's run. The game will run forever, and has no glitches or bugs, making it resemble a semi-professional software.

Overall, we think this is a beautiful project that we should totally get A+ on.