Alicia Prince, Aryan Mitharwal, Hadya Rohin, Hoang Quan Dinh, Karlos De Los Santos

Professor Qiacho Dong

CS 2400

3 May 2024

Socially Responsible Computing Project

We created an in-house inventory management system for our Socially Responsible Computing project to help the farm track its produce. The farm currently uses a paper-pencil tracking system to manage its donations and produce, and we want to help them change that. Our main object was to create a reliable and efficient inventory management system for the local farm. Our group wanted to include a way to sort the different produce by quantity and weight. This would help them keep track of donations and the amount of produce they grow each month. It could also provide some statistical insight into what is most taken from the donation bin and what produce they should focus on maximizing to help the community.

Our inventory management system is meticulously designed with user-friendliness as the cornerstone. We are developing several vital features that will simplify the process of inputting produce types, quantities, and weights and removing items. We aim to create a system that is efficient and a breeze to manage and maintain. The system's structure revolves around three core classes, each serving a specific function to ensure smooth operation and a delightful user experience.

The data structure we are utilizing is the ADT bag. We’re utilizing a bag because it is the easiest to implement, and for what is needed for our data structure, which is storing, adding, and removing specific items without regard to order, the ADT bag is the most fitting. For the implementation of the bag, the ArrayList is used because we desire for our storage to be able to store any number of objects, which the ArrayList achieves, and also because it’s already implemented. It should be noted that while we’re technically using a List, implementing our methods doesn’t specify position or order, so in function, it’s more like the ADT bag than the ADT list.

The Main class is the heart of our inventory management system. It serves as the entry point and central control hub, managing the interaction between user interface components, inventory data, and file I/O operations. Upon initialization, the Inventory instance is created and populated with data from the 'inventory.txt' file. A shutdown hook is registered to ensure data persistence across sessions. The file is written in CSV format, which is simple to understand and doesn’t require much storage. This class is crucial to the system's operation, ensuring a seamless and efficient user experience.

The GUI components, such as buttons, text fields, and text areas, are designed to make it easy for users to interact with the system. These components are arranged within panels and added to the JFrame to create a unified user interface. The ActionListener interfaces are used to handle user actions, such as searching for products, adding or removing products, and displaying product details. These actions trigger corresponding methods in the Inventory class, allowing for seamless interaction with the underlying data. Overall, the Main class is pivotal in facilitating user interaction with the inventory system, providing a streamlined and intuitive interface for effectively managing farm produce inventory.

The Produce class contains various setters and getters necessary for our program to function and execute. It holds details like name, weight, and quantity for each item. You can create a new item using the constructor, and there are methods to get and set these details. The toString() method turns the details into a simple string, separated by commas and following the CSV format, while the toStringPretty() method makes it look friendly and organized. It's the backbone for managing produce in our inventory system.

The Inventory class in our code plays a central role in managing our program. This class holds the ArrayList, where the Produce objects would be added to/ removed from. It also contains most of the methods we used to operate our inventory system. Some essential methods that we implemented are addProduce, removeProduce, addToQuantity, removeQuantity, addToExistingEntry, searchForProduce, and showProduceData. The addProduce and removeProduce methods allow users to add or remove different types of produce as per their requirements. Other methods, like addToQuantity, removeQuantity, and addToExistingEntry, enabled users to modify the previously entered data into the inventory. The searchForProduce and showProduceData methods were designed to help users search for different types of produce and display their data, such as weight and quantity. The method gets called using a critical listener on our search bar; once the user types the product they want to search for, they must hit enter for the results to show up.

Java Unit testing was implemented to locate and debug errors efficiently. The approach behind the test was to isolate each function to detect where the errors originated. The actual return from the function is compared against the expected value to ensure accuracy and proper functionality. Once the functions are isolated, Java Unit testing executes each function individually. It compares the actual output of the function with the expected output, as defined by the test case. If there's a discrepancy between the actual and expected values, it indicates a potential error in the function. This process helps pinpoint and debug errors efficiently, allowing for accurate and reliable program functionality.

While our code was successful, and we were able to solve the problem that we were hoping to solve, there were also some improvements that we could make to our program. We had initially planned to make an SQL database. However, due to time constraints, we were not able to implement the database. We also could have made a system that tracks our donated produce and connects the two systems so that adding a specific type of produce to the donated section automatically subtracts from the produce quantity. Many improvements could have also been made if we had more time. Those include having a security aspect so that only the people on the farm can access it, using a barcode system for the farm to be able to scan barcodes rather than typing out the produce, and including a warning message when produce is running low or is out. These would all help elevate the program's stability and create a more significant, more straightforward experience for the farm.

Overall, our project was made to solve the issue of inventory management. We hoped to provide them with a more efficient, organized, and seamless method. We wanted to incorporate a technique that would minimize human error and allow them to rely on the most accurate data. Through the program's framework, the variety of classes, the methods, and more, we created a solution and provided a better opportunity for the farm to grow and improve. Given our limitations, we made sure to fix any errors and test our program to ensure that it was the best version. We were able to implement aspects such as GUI, which provided a more accessible experience for the user. The simplicity adds to our program in that it allows the farm to utilize our program without it being too much of a hassle or more of a burden to them. We were able to implement many aspects of what we learned to help create this program, and while the program is usable, there were also aspects of the program that could have been improved. Despite this, the program resolved the issue we had seen on the farm and provided them with a step in a more modern and efficient system that would positively impact their farm.