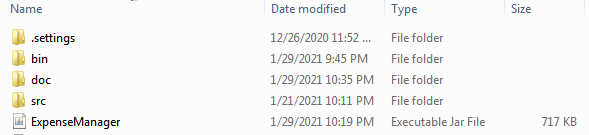
**Criterion C: Development**

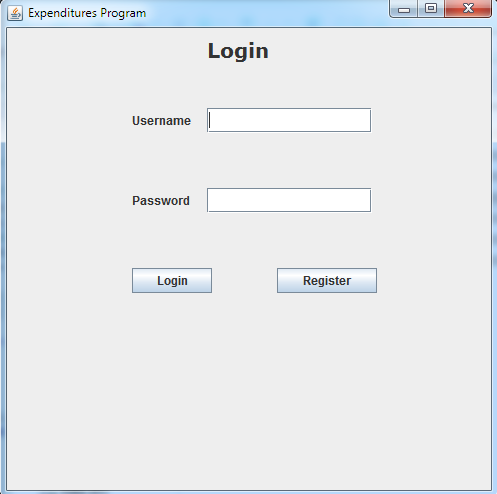
The product is a Java program, written in the Eclipse IDE. The program’s functionality allows a user to create and register an account to access the program. The user can then select an existing spreadsheet and make modifications or create an entirely new spreadsheet. All inputted data should be saved in text files and the spreadsheet is saved in an Excel file.

The program is executable by clicking the “ExpenseManager.jar” file, (Figure 1) and a login screen will be displayed (Figure 2).



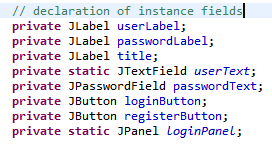
*Figure 1: Executable JAR file.*

Login Window:



*Figure 2: Login Screen*

For the graphical user interface, I used javax.swing including JFrames, JPanels, JButtons, JLabels, JTextFields, and JPasswordFields in the entire application because it is easy and efficient (Figure 3).



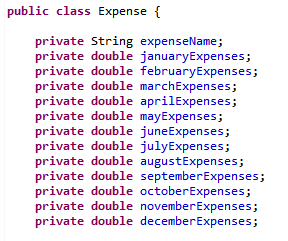
*Figure 3: javax.swing objects in the Login class*

**Computational Techniques and Algorithms:**

1. Encapsulation:

Contrary to my design in Criterion B, my client asked for a more practical interface when adding expenses. Instead of using income and outcome, and the specification of dates, my client proposed that I sort the expense by month, hence I created the Expense class.

I used encapsulation in the Expense class (Figure 4) because this ensured inputted values by the user are valid for the spreadsheet and more than one instance of the Expense can be run at the same time. The private field in the Expense class, including the expenseName, can only be modified by the public void setExpenseName(String name) (Figure 5) method, and accessed through the public double getExpenseName() method (Figure 6). This method was used in the EditExpense class (Figure 7).



*Figure 4: Encapsulated fields of Expense class*



*Figure 5: Sample of encapsulated modifier method in class Expense*



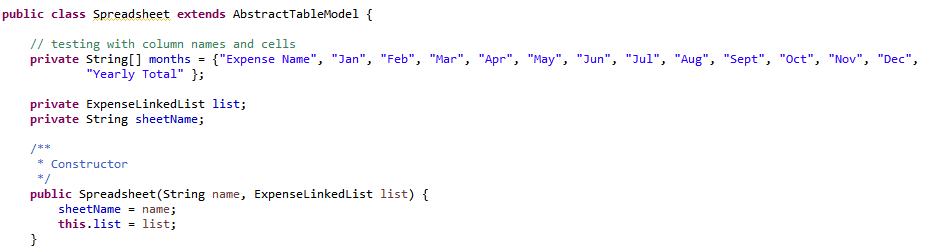
*Figure 6: Sample of encapsulated accessor method in class Expense.*



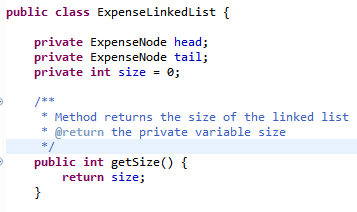
*Figure 7: Use of encapsulated modifier method in class EditExpense*

2. LinkedList:

I created a custom singly linked list class, called ExpenseLinkedList (Figure 9), with nodes defined by the class ExpenseNode (Figure 10), which holds objects of type Expense. I used this abstract data type over an array because the number of expenses that can be stored is not constant. When the user clicks “Add an Expense” or “Delete an Expense”, the Spreadsheet class (Figure 8) will be notified and perform its appropriate action by calling methods in the ExpenseLinkedList class. These methods will be explained in Table 1.



*Figure 8: Spreadsheet class taking in an ExpenseLinkedList as a parameter*



*Figure 9: Encapsulated instance fields of ExpenseNode in ExpenseLinkedList*



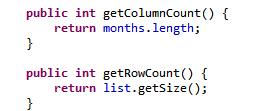
*Figure 10: Encapsulated variables and methods of ExpenseNode.*

*Table 1: Methods of the ExpenseLinkedList class*

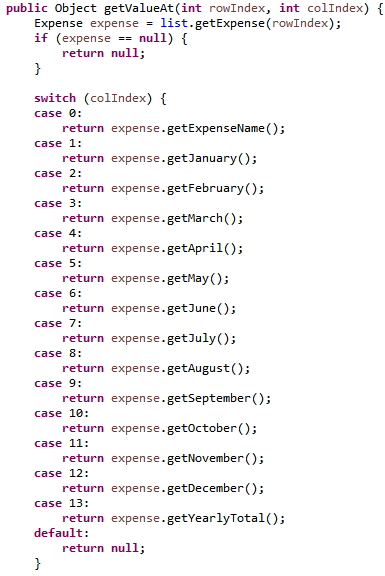
|  |  |  |
| --- | --- | --- |
| **Method** | **How it works** | **Practicality in program** |
| *Figure 11: addBack(Expense expense) method* | * Expense is added to end of linked list * Node is now “tail” of linked list | The expense is displayed at the bottom of the spreadsheet. |
| *Figure 12: deleteAt(int i, Expense expense) method* | * ExpenseNode at user’s specified position is deleted from linked list * Linked list must change node in front of it as original node was removed | The user can select any expense and remove them easily. The information of the expense will deleted |
| *Figure 13: removeFront() method* | * Head of linked list is removed by unlinking current head of linked list * Sets the new head as the next ExpenseNode * Returns original head | If the user selects to remove the first expense in the spreadsheet, the expense will not be stored in the linked list and the row will be removed. |
| *Figure 14 The getExpense(int i) method* | * Traverse through linked list until index of ExpenseNode matches with index i * Returns details of expense by calling getExpense() method in ExpenseNode class * Asserts false if caller asks for “non-existent element” (Geeksforgeeks .com, 2020) | If the user selects an expense in the table to edit, the user will be able to see the data stored for that expense in the EditExpense JFrame. |

3. Inheritance:

In my Spreadsheet class, I extended Java’s built-in AbstractTableModel because I wanted to create my custom JTable. The AbstractTableModel is an “abstract class that provides default implementations for most of the methods in the TableModel interface”. (Docs.Oracle.com, 2021). By using the inherited methods of getRowCount(), getColumnCount(), and getValueAt(int row, int column) (Figure 15, 16), I was able to create my own version of a JTable by accessing the number of rows and columns, and retrieving values by using a switch case algorithm to retrieve values for each expense.



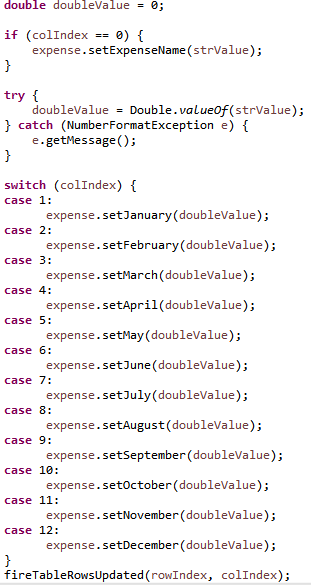
*Figure 15: Sample inherited methods from the AbstractTableModel class.*



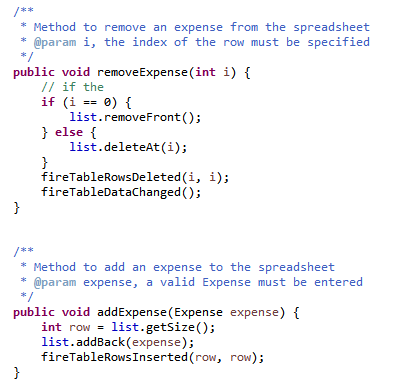
*Figure 16: Inherited getValueAt(int rowIndex, int colIndex) from AbstractDataModel class*

Although the methods outlined in Figures 15 and 16 are the only three methods required for a “concrete TableModel”, I included other inherited methods such as the setValueAt(Object value, int rowIndex, int colIndex) to modify the contents of an expense (Figure 17) (Docs.Oracle.com, 2021). However, some methods were created including addExpense(Expense expense) and removeExpense (Figure 18) in the class to suit the success criteria of adding and deleting expenses in the spreadsheet.

To update the contents of the JTable, the inherited methods of fireTableRowsInserted(), fireTableRowsDeleted(), and fireTableDataChanged() are called (Figures 17 and 18)

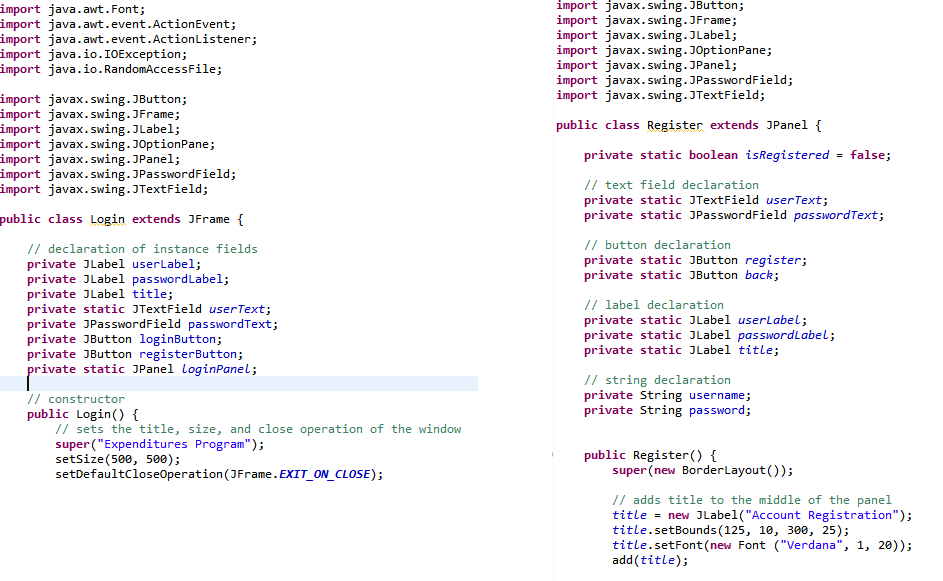


*Figure 17: Body of setValueAt(Object value, int rowIndex, int colIndex) method in Spreadsheet class*



*Figure 18: removeExpense(int i) and addExpense(Expense expense) methods in the Spreadsheet class*

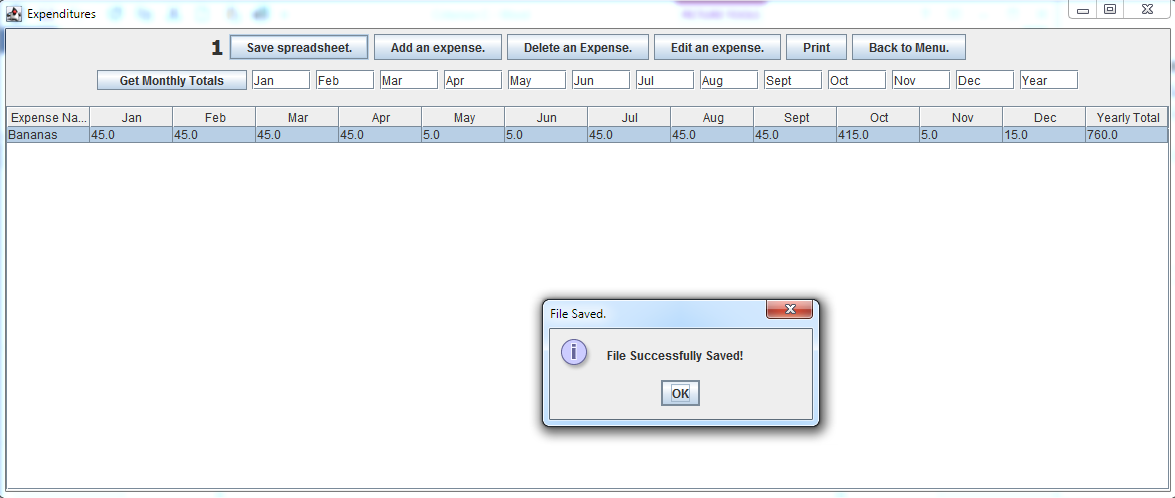
Other inherited methods came from JFrames and JPanels for classes, (e.g. Login and Register respectively), shown in Figure 19. These methods were used to set the basic layout and close operations of the window and panel.



*Figure 19: Login and Register class that extends JFrame and JPanel respectively, inheriting methods from its superclass.*

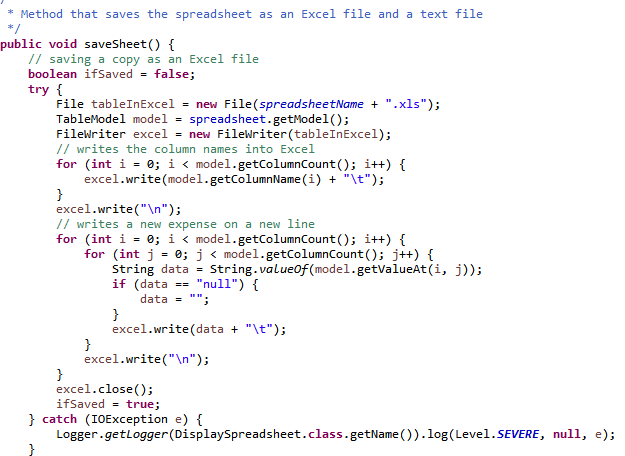
4. Saving Files

One of the success criteria of the project was to ensure that the data from the table is saved. I created a method to export the JTable as an Excel file (.xlsx). Once the spreadsheet is saved, the user will be notified that it was saved by a pop-up message (Figure 20).



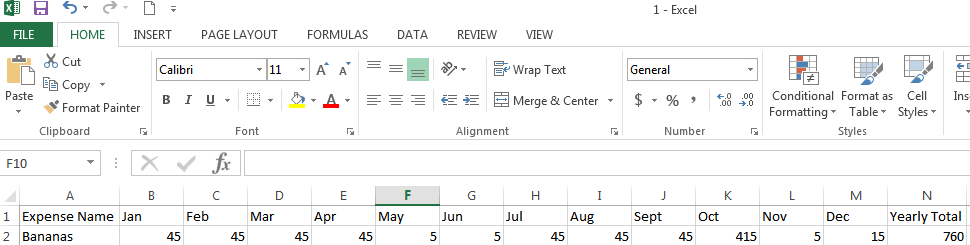
*Figure 20: Pop-up message after saving the spreadsheet.*

This was created in the DisplaySpreadsheet class by calling the method saveSheet(). It implements a “for loop” to first name the column headers, and then uses a nested “for loop” to save the contents of each expense into a cell in the Excel file (Figure 21).



*Figure 21: saveSheet() method in DisplaySpreadsheet class*

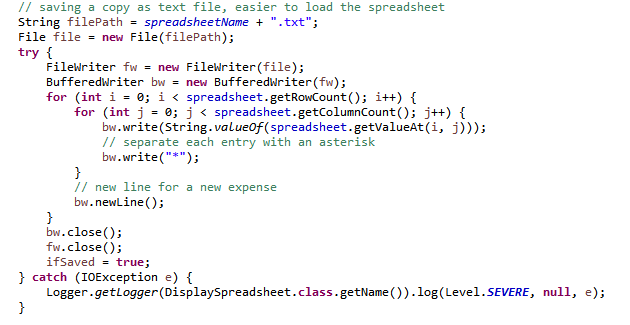
Once the user opens the Excel file, the contents of each expense is displayed in the order the user saved in the program (Figure 22).



*Figure 22: Sample Excel file of saved spreadsheet.*

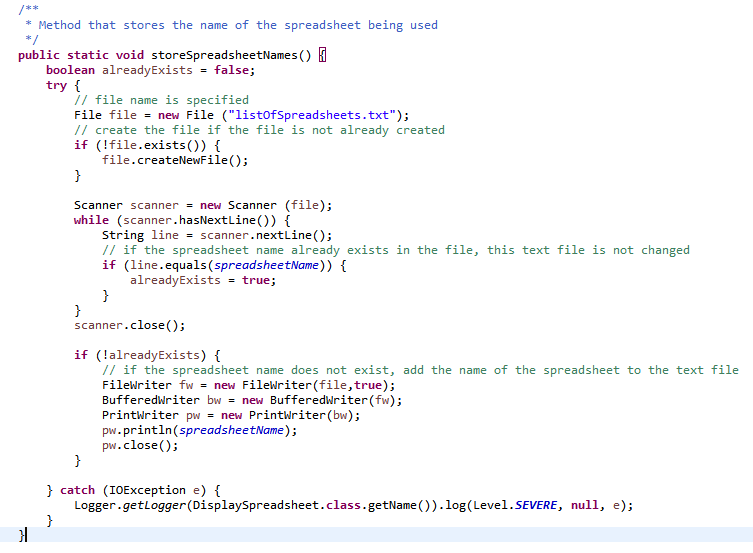
5. Loading Spreadsheets:

Not only does the saveSheet() method save the user’s spreadsheet as an .xlsx file, a text file is created with the table contents. I chose to create a separate text file because it would be easier to load the spreadsheet back into the table, meeting another point in the project’s success criteria. A FileWriter and a BufferedWriter is used to write down the contents of the spreadsheet, with an asterisk separating the value of each cell (Figure 23). This code was adapted from the Stack Overflow community.



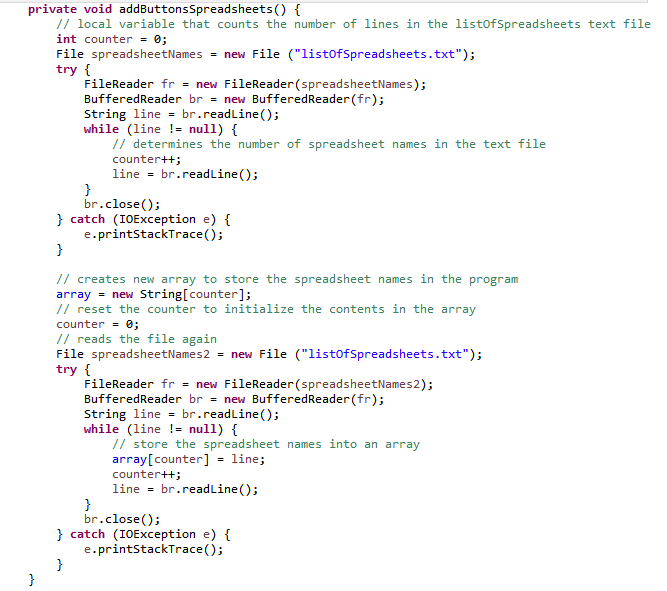
*Figure 23: Converting the spreadsheet into a text file*

Next, the name of the spreadsheet is stored in separate text file called listOfSpreadsheets.txt. This contains the names of all existing spreadsheets with no overlap in names to avoid confusion in loading (Figure 24).



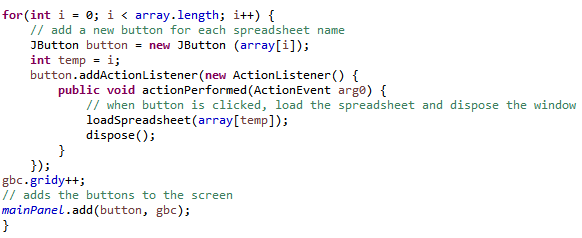
*Figure 24: storeSpreadsheetsNames() method in the DisplaySpreadsheet class stores the name of every spreadsheet created on a separate line*

This text file will be parsed through during in the SpreadsheetSelect class and initialize a String array containing these names (Figure 25).



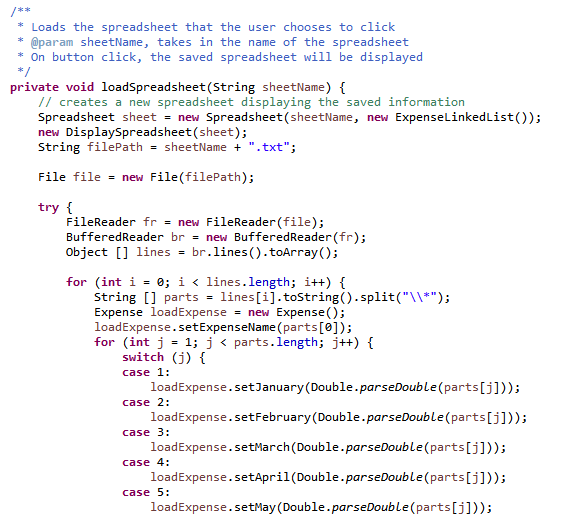
*Figure 25: Private method addButtonsSpreadsheets() in the SpreadsheetSelect class*

Each saved spreadsheet will have its own button, and added to the main panel. When clicked, the private method loadSpreadsheet(String name) is called, and takes in the name of the spreadsheet as a parameter. (Figure 26).



*Figure 26: Adding a button for each spreadsheet in the SpreadsheetSelect class*

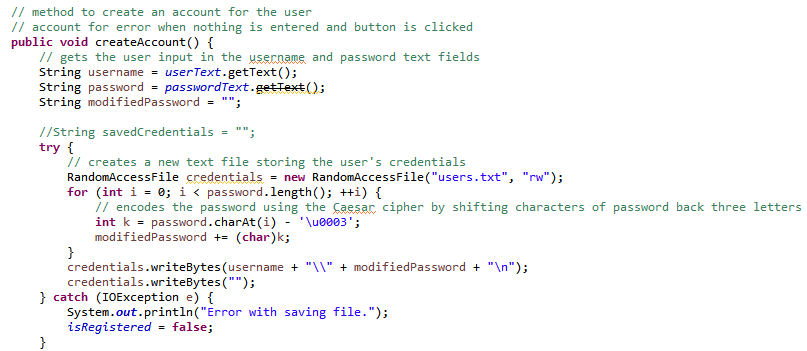
This calls the saved text file of the spreadsheet, reads the contents of the table through an array, and sets the values by calling on the various modifier methods in the Expense class. Ultimately, a “new” DisplaySpreadsheet() class is created and returns the saved spreadsheet (Figure 27).



*Figure 27: Sample of method loadSpreadsheet(String name) in the SpreadsheetSelect class*

6. Basic Encryption

When the user registers for an account, their credentials are saved as a text file and the password is encrypted using the Caesar cipher to promote security. When the user logs in, the saved password is decrypted and compared to the user input to determine the user’s correct credentials (Figure 28).

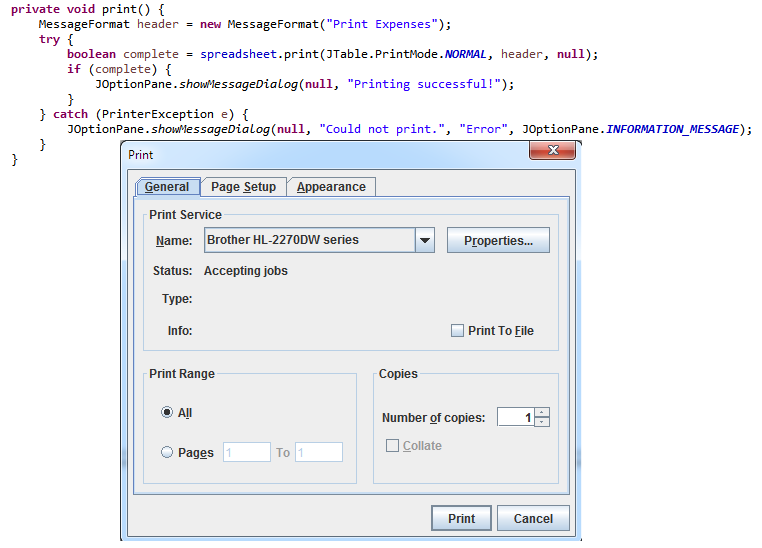




*Figure 28: createAccount() method from the Register class and checkCredentials() method from the Login class*

7. Print

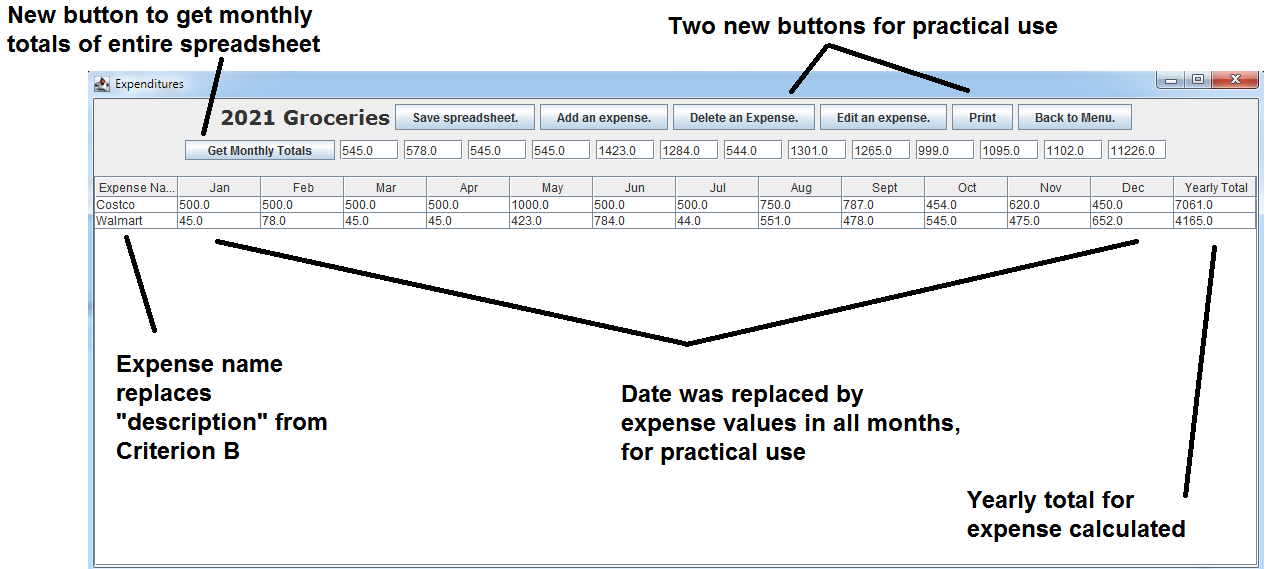
Upon further discussion with my client, a suggestion was to include an option to print the spreadsheet. Therefore, I created the print button where the program will try to print the current list of expenses on button click. The code below was adapted from Docs.oracle.com. (Figure 29).



*Figure 29: print() method in DisplaySpreadsheet*

**Changes to Design:**

My final program has incorporated many changes compared to its initial design because of feedback from the ICS teacher and my client (see Appendix D). The initial design can be found in Criterion B and Figure 30 represents the new design.

**

*Figure 30: Sample view of spreadsheet program with annotations describing new changes*

**Total words in Criterion C:** 966 words (excluding figure captions, references, and tables)

**Works Cited:**

Docs.Oracle.Com. "How To Print Tables (The Java™ Tutorials > Creating A GUI With JFC/Swing > Using Other Swing Features) ". Docs.Oracle.Com, 2021, https://docs.oracle.com/javase/tutorial/uiswing/misc/printtable.html. Accessed 29 Jan 2021.

Geeksforgeeks. "Write A Function To Get Nth Node In A Linked List". Geeksforgeeks, 2009, https://www.geeksforgeeks.org/write-a-function-to-get-nth-node-in-a-linked-list/. Accessed 29 Jan 2021.

StackOverflow Community, “Export JTable in Excel file”, Stack Overflow, 2014, https://stackoverflow.com/questions/22560566/export-jtable-in-excel-file. Accessed 30 Jan 2021.