This Java code represents an application called "TheDesk" that allows users to manage their expenses. It provides various options for reviewing, adding, deleting, sorting, and searching expenses. Here's a breakdown of the code:

1. The code starts with importing necessary Java libraries: ArrayList, Collections, InputMismatchException, and Scanner.

2. The Main class is defined with a private static ArrayList variable named expenses, which will store the user's expenses.

3. The main method is the entry point of the program. It displays a welcome message and calls the optionsSelection method.

4. The optionsSelection method presents a menu of available options to the user and handles their choice. The menu options are stored in an array of strings. The method iterates through the array and prints each option. Then, it prompts the user to enter their choice.

5. The user's choice is read using a Scanner object. The code then uses a switch statement to perform different actions based on the chosen option.

6. If the user selects option 1, it displays the list of saved expenses by printing the expenses ArrayList. After that, it calls optionsSelection again to display the menu.

7. If the user selects option 2, they are prompted to enter the value of the expense they want to add. The entered value is read using a Scanner and then added to the expenses ArrayList. The updated list is printed, and optionsSelection is called again.

8. If the user selects option 3, they are warned about deleting all expenses and asked to confirm by selecting the same option again. If the confirmation matches the selected option, the expenses ArrayList is cleared, and an appropriate message is displayed. If the confirmation fails, an error message is shown. Finally, optionsSelection is called again.

9. If the user selects option 4, the sortExpenses method is called to sort the expenses ArrayList in ascending order. The sorted list is printed, and optionsSelection is called again.

10. If the user selects option 5, they are prompted to enter the expense they want to search for. The entered value is read using a separate Scanner object, and the searchExpenses method is called to check if the expense is present in the expenses ArrayList. The search result is displayed, and optionsSelection is called again.

11. If the user selects option 6, the closeApp method is called, which displays a closing message and terminates the program.

12. If the user selects an invalid option, an error message is displayed, and optionsSelection is called again.

13. The closeApp method simply prints a closing message.

14. The searchExpenses method takes an ArrayList and an expense value as parameters. It iterates through the ArrayList and checks if the expense value is present. If found, it displays a message indicating the expense was found; otherwise, it displays a message indicating it was not found.

15. The sortExpenses method takes an ArrayList as a parameter and uses the Collections.sort method to sort the ArrayList in ascending order. The sorted list is then printed.

Overall, this code provides a simple command-line interface for managing expenses, allowing users to perform various operations on their expenses such as adding, deleting, searching, and sorting them.

Steps and algorithms used to fix the bugs:-

1. Downloaded the code from github.
2. Ran the code once to understand the flow.
3. Added searching logic by using linear search algorithm.
4. Added sorting logic by using sort method from collections class.
5. Fixed logic in adding expenses code.
6. Removed unnecessary code. (arr1, arrlist, for loop and if statement enclosing switch)
7. Added exception handling logic to check for input mismatch error.