Title: The Desk Expense Manager

Introduction:

The Desk Expense Manager is a Java-based application designed to help users manage their expenses efficiently. It provides various functionalities such as reviewing, adding, deleting, sorting, and searching for expenses.

Features:

1. **Review Expenses:** Users can review their saved expenses, which are displayed in the console. The program retrieves expenses from an ArrayList and prints them out for the user.
2. **Add Expenses:** Users can add new expenses to their list. The program prompts the user to enter the value of the expense, which is then added to the list of expenses.
3. **Delete Expenses:** Users can delete all their expenses at once. The program prompts the user for confirmation before erasing all expenses from the list.
4. **Sort Expenses:** Users can sort their expenses in ascending order. The program utilizes the Collections.sort() method to arrange expenses from the lowest value to the highest.
5. **Search Expenses:** Users can search for a specific expense from their list. The program prompts the user to enter the expense they want to search for, and it checks if the expense exists in the list.
6. **Close Application:** Users can close the application when they are done managing their expenses. Upon closing, the program displays a closing message.

Implementation:

The application is implemented using Java programming language. It utilizes an ArrayList to store the expenses entered by the user. The main method controls the flow of the program by displaying options to the user and invoking corresponding methods based on user input.

Conclusion:

TheDesk Expense Manager provides a simple yet effective solution for managing expenses. With its user-friendly interface and essential features, users can easily keep track of their spending. Whether it's reviewing past expenses, adding new ones, or sorting/searching for specific expenses, this application offers a convenient way to manage finances.

Algorithm: -

1. **Initialization:**
    - The program initializes an ArrayList to store the user's expenses.
    - It displays a welcome message and presents options to the user.
2. **Option Selection:**
    - The program prompts the user to select an option (1-6) from the menu.
    - It reads the user's choice using a Scanner object.
3. **Option Execution:**
    - Based on the user's choice, the program executes the corresponding functionality.
    - If the user chooses to review expenses, the program displays the list of expenses.
    - If the user chooses to add expenses, the program prompts for the expense value and adds it to the list.
    - If the user chooses to delete expenses, the program confirms the action and clears the expense list.
    - If the user chooses to sort expenses, the program sorts the expenses in ascending order.
    - If the user chooses to search expenses, the program prompts for the expense to search and checks if it exists in the list.
    - If the user chooses to close the application, the program terminates.
4. **Recursion:**

- After executing the chosen functionality, the program recursively calls the optionsSelection() method to display the menu again.
- This continues until the user chooses to close the application.

5. **Search and Sort Methods:**
    - The searchExpenses() method prompts the user for an expense value to search and checks if it exists in the expense list.
    - The sortExpenses() method uses Collections.sort() to sort the expenses in ascending order.
6. **Closing the Application:**
    - When the user chooses to close the application, the program displays a closing message and terminates.

Overall, the algorithm follows a menu-driven approach where the user selects options to perform various operations on their expenses. The program utilizes recursion for the menu loop and includes methods for searching, sorting, and modifying the expense list based on user input

<u>Bug Fixes: -</u>

The previous code contained several bugs and areas for improvement, including:

1. **Redundant Loop:**
    - The code included a loop for options validation that was unnecessary and led to redundant checks.
2. **Inefficient Menu Handling:**
    - The code used a loop to iterate over options, which complicated the menu handling process.
    - It could be simplified by using a switch-case statement directly to handle user input options.
3. **Incomplete Methods:**
    - The **searchExpenses()** and **sortExpenses()** methods were declared but left incomplete, leading to compilation errors.
    - These methods needed to be implemented to fulfill their respective functionalities.
4. **Minor Corrections:**
    - Some print statements required minor corrections for better clarity and consistency.

To address these issues and improve the code, the following changes were made:

1. **Removed Redundant Loop:**
    - The unnecessary loop for options validation was removed, simplifying the code structure.
2. **Simplified Menu Handling:**
    - The switch-case statement was directly used to handle user input options, making the menu handling more straightforward.
3. **Implemented Incomplete Methods:**
    - The **searchExpenses()** method was completed to prompt the user for an expense value to search and check if it exists in the expense list.
    - The **sortExpenses()** method was implemented to sort the expenses in ascending order using **Collections.sort()**.
4. **Corrected Print Statements:**
    - Minor corrections were made in some print statements to improve clarity and consistency in the output messages.

These changes addressed the bugs and inefficiencies in the previous code, resulting in a more functional and robust Expense Manager application.