

CS 520: Homework 3 - Implementation & Debugging

1. Objective:

The feature focuses on enabling users to export their list of transactions to a CSV file. Care has been taken that the filename is valid and is formatted properly

2. Functionality Description:

- The user is asked to enter the filename that they want to export
- The entered filename is validated
- Once the validity of the filename is done, the transactions are exported to the file in CSV format
- The first row of the file has headers → Date, Amount, Category
- Every row after the header has a transaction, with each row properly formatted

3. Architecture Description

- The export feature in the application is designed using MVC (Model View Controller) design pattern.
- This separates the concern, which will be explained in the coming part
- As per our design,
The **Model** manages the list of transactions.
The **View** helps in the interaction with the user and takes the input and displays the messages that let the user know if the file has been exported successfully
The **Controller** validates the filename and does the actual export process
- This architecture helps in maintaining modularity, thus making it easy to troubleshoot and easy to maintain

4. Brief Description of Model, View, and Controller

a. Model:

- The model maintains and manages the list of all transactions
- It provides access to the stored transactions
- These transactions can then be exported.

- The model also handles data-related logic such as adding, removing, and retrieving transaction records.
- The existing getTransactions method of the model can be used to get list of transactions in read-only format for the export operation.

```

    /**
     * Returns unmodifiable list of all transactions.
     * @return List of transactions
     */
    public List<Transaction> getTransactions() {
        return Collections.unmodifiableList(new ArrayList<>(transactions));
    }

```

- The unmodifiable list that is returned helps in maintaining the integrity of the data

b. **View:**

- The View interacts with the users and prompts for input by asking for a filename
- It also displays messages that let the user know if the export process has been successful or not
- A method, as shown below, can be added to ask the user for a filename

Java

```

public String provide_FileName() {
    return JOptionPane.showInputDialog(null, "Hello, User! Please enter
your filename");
}

```

- Similarly, another method can be used to display the messages that confirm to the user if the file has been successfully exported or if the export has failed.

Java

```

public void show_message(String message) {
    System.out.println(message); // Outputs messages to user
}

```

c. **Controller:**

- The Controller is responsible for handling data for exporting transactions
 - It is also responsible for validating the filename provided by the user
 - Additionally, it coordinates with the view to display messages and with the model to retrieve transactions
 - It initiates the process to write transactions to the file as requested by the user
 - To facilitate this process, the following methods are used
 - `export_file_to_CSV()` for actual import logic
 - `make_header_valid()` is used to remove comma so that each column name looks properly formatted.
- It checks if the filename has csv. It also checks for special characters as well as for the empty or null
- `check_if_file_valid()` to check the filename provided by the user is correct or not

Java

```

private static final String FILE_HEADERS = "Date,Amount,Category";
private static final String FILE_NAME_INVALID = "Dear User, Entered file name
is invalid. It must end with .csv and not contain special characters.";
private static final String SUCCESSFUL_EXPORT = "Dear User, Transactions are
successfully exported to the given file";
private static final String ERROR_IN_EXPORT = "Dear User, Something went wrong
while exporting: ";

public void export_file_to_CSV(String name_file) {
    if (!check_if_file_valid(name_file)) {
        view.displayMessage(FILE_NAME_INVALID);
        return;
    }

    try (FileWriter f_writer = new FileWriter(name_file)) {
        f_writer.write(FILE_HEADERS + "\n");
        for (Transaction w_tran : model.getTransactions()) {
            writer.write(String.format("\"%s\", \"%s\", \"%s\", %.2f\n",
                make_header_valid(w_tran.getDate()),
                make_header_valid(w_tran.getTimestamp()),
                w_tran.getAmount()));
        }
        view.displayMessage(SUCCESSFUL_EXPORT + ": " + name_file);
    } catch (IOException e) {
        view.displayMessage(ERROR_IN_EXPORT + e.getMessage());
    }
}

```

```
}
```

```
Java
```

```
private String make_header_valid(String value) {
    return value.replace("\n", "\n\n");
}
```

```
Java
```

```
private boolean check_if_file_valid(String name_file) {
    if (name_file == null || name_file.trim().isEmpty()) return false;
    if (!name_file.toLowerCase().endsWith(".csv")) return false;
    if (name_file.matches(".*[\\\\\\\\\\\\:\\*?\"<>|].*")) return false;

    String f_name = name_file.replaceAll("(?i)\\.csv$", "").toUpperCase();
    boolean valid_Name =
    !f_name.matches("CON|PRN|AUX|NUL|COM[1-9]|LPT[1-9]");
    return valid_Name;
}
```

5. UI Design Laws

1. Interactive:

- User centric interaction is an important aspect of user interface design

- When the error occurs, the user must not be left wondering what might have gone wrong
- The user must be provided with the information regarding the success and failure of the task that they are trying to complete
- Our system lets the user know when the entered filename is incorrect

2. **Provides help:**

- When the user inputs filename that doesn't abide by the validations for it, our design provides error message which is concise and clear
- This way the user can make the necessary changes and achieve the task successfully.
- Such hints help in reducing the confusion and saves the user the frustration.

Example:

When the provided filename is in incorrect format, the user sees error message like -

"Entered file name is invalid. It must end with .csv and not contain special characters."

6. **OO Design principles:**

1. Single Responsibility principle:

- As per this principle a class should have one and only one reason to change
- In our design, we have separated the responsibilities so that Controller is only responsible for functionalities like
- `export_file_to_csv` is responsible only for the export logic.
- `check_if_file_valid` is solely responsible for validating file names.
- `make_header_valid` is responsible for escaping/cleaning individual fields
- Model class is responsible for managing data (transactions)
- View class handles messaging

Thus, it can be seen that each of the classes has a focussed responsibility, thus obeys this principle

2. Open/Closed Principle:

- This principle states that system should be open for extension but closed for modification

- In our design, we can extend the logic for validating the filename like adding checks to set a limit to the size of the filename
- Similarly, currently, we only want the filename to be csv. However, this can be extended to include other formats as well like JSON, XML
- Hence, our system can be extended without changing the core

3. Dependency Inversion Principle

- This principle states that our system should depend on abstractions and not on concretions
- It is suggested in our design where Model and View are injected into the controller i.e. the controller doesn't instantiate them directly.
- Thus, this principle is implemented because we have avoided tightly coupling between controller and implementation details

7. Best Practices:

1. Using Method and Variable names that are self-explanatory:

- The method and variable name clearly explain what they do without any confusion
- This makes our code self explanatory

2. Encapsulation and Information Hiding:

- We are using unmodifiable list of transactions that we access from model using method getTransactions()
- Thus, this avoids accidental modification
- Hence, our Internal data is protected and only exposed through controlled access

3. Avoiding Magic Strings:

- We have used constants for headers in the csv file
- The error messages are also substituted with a constant
- This makes the code easy to modify where just change a single constant would help in modifying all the places where it is used
- This improves the readability and maintainability of the code

Java

```
private static final String FILE_HEADERS = "Date,Amount,Category";
```

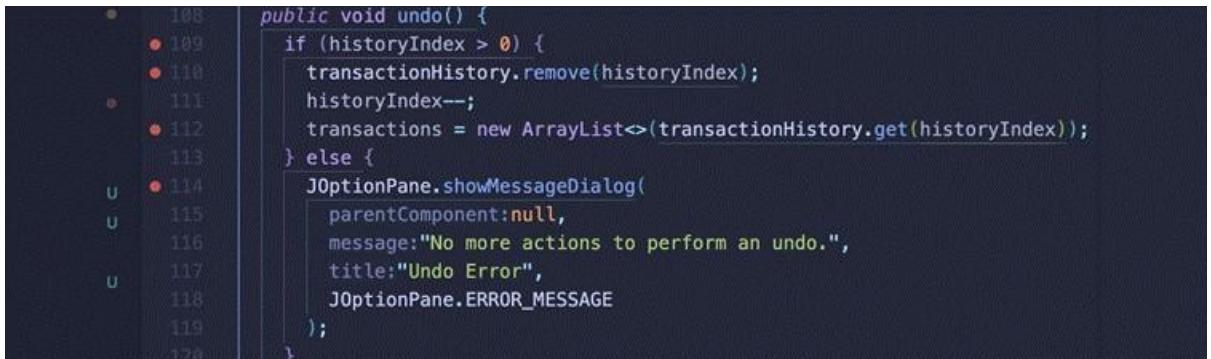
```
private static final String FILE_NAME_INVALID = "Dear User, Entered file name  
is invalid. It must end with .csv and not contain special characters.";  
private static final String SUCCESSFUL_EXPORT = "Dear User, Transactions are  
successfully exported to the given file";  
private static final String ERROR_IN_EXPORT = "Dear User, Something went wrong  
while exporting: ";
```

4. Error Handling:

- We don't want the app to shut down because of the error
- Hence, it is very important that the user is made aware of the potential error so that the user can mitigate it by making the necessary modifications.
- In our design, error is handled properly with the help of proper try and catch and also relevant error messages

Debuggability.

Debuggability - IDE showing the breakpoint for the undo

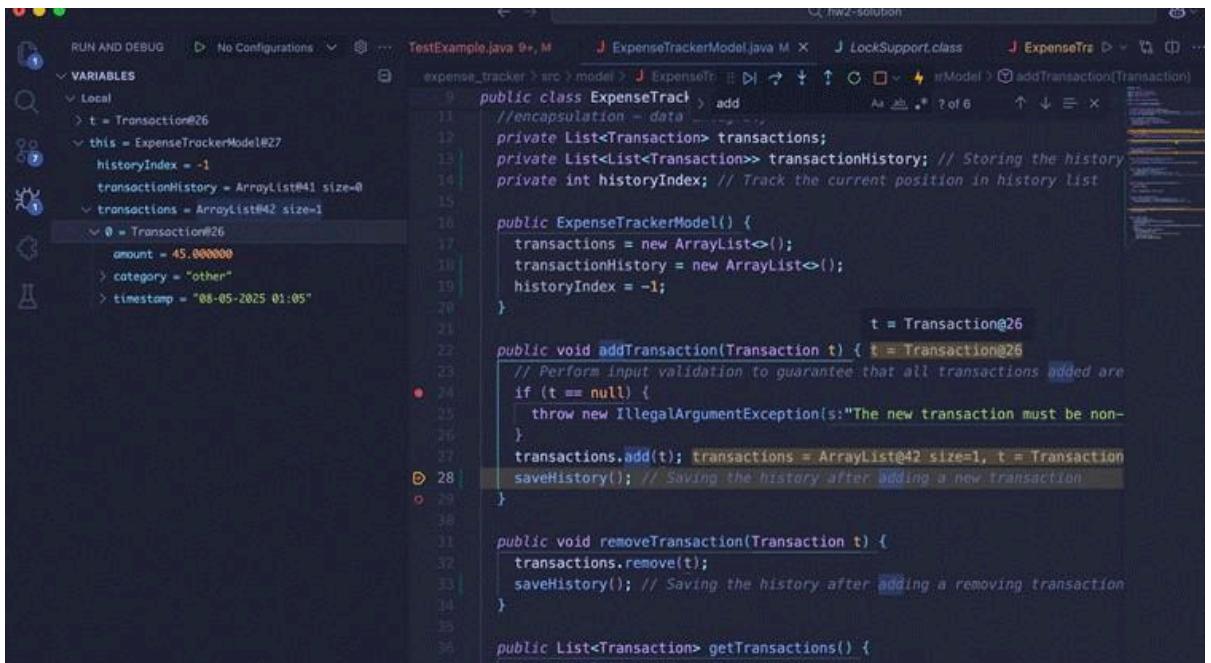


A screenshot of an IDE showing the code for the `undo()` method. The code is as follows:

```
public void undo() {
    if (historyIndex > 0) {
        transactionHistory.remove(historyIndex);
        historyIndex--;
        transactions = new ArrayList<>(transactionHistory.get(historyIndex));
    } else {
        JOptionPane.showMessageDialog(
            parentComponent:null,
            message:"No more actions to perform an undo.",
            title:"Undo Error",
            JOptionPane.ERROR_MESSAGE
        );
    }
}
```

The line `transactionHistory.remove(historyIndex);` is highlighted with a red rectangle, indicating it is the current line of execution or has a breakpoint set. The code editor shows line numbers from 108 to 120.

Debuggability - Debugger showing the program execution state(usually the variables view)after calling addtransaction but before calling undo (show the model and/or UI widgets set appropriately)



A screenshot of an IDE showing the debugger's variables view and the code for the `addTransaction` method. The variables view shows:

- `t = Transaction@26`
- `this = ExpenseTrackerModel@27`
- `historyIndex = -1`
- `transactionHistory = ArrayList@41 size=0`
- `transactions = ArrayList@42 size=1`
 - `0 = Transaction@26`
 - `amount = 45.000000`
 - `category = "other"`
 - `timestamp = "08-05-2025 01:05"`

The code editor shows the `addTransaction` method with a red dot at line 24, indicating the current execution point. The code is as follows:

```
public class ExpenseTracker {
    // Encapsulation - data
    private List<Transaction> transactions;
    private List<List<Transaction>> transactionHistory; // Storing the history
    private int historyIndex; // Track the current position in history list

    public ExpenseTrackerModel() {
        transactions = new ArrayList<>();
        transactionHistory = new ArrayList<>();
        historyIndex = -1;
    }

    public void addTransaction(Transaction t) { t = Transaction@26
        // Perform input validation to guarantee that all transactions added are
        if (t == null) {
            throw new IllegalArgumentException("The new transaction must be non-
        }
        transactions.add(t); transactions = ArrayList@42 size=1, t = Transaction
        saveHistory(); // Saving the history after adding a new transaction
    }

    public void removeTransaction(Transaction t) {
        transactions.remove(t);
        saveHistory(); // Saving the history after removing a transaction
    }

    public List<Transaction> getTransactions() {
    }
}
```

Added new transaction and the debugger entered the `addTransaction` block

```
public class ExpenseTracker {
    private ExpenseTrackerModel model = new ExpenseTrackerModel();
    private List<Transaction> transactions;
    private List<List<Transaction>> transactionHistory;
    private int historyIndex;

    public ExpenseTracker() {
        transactions = new ArrayList<>();
        transactionHistory = new ArrayList<>();
        historyIndex = -1;
    }

    public void addTransaction(Transaction t) {
        if (t == null) {
            throw new IllegalArgumentException("The new transaction must be non-null");
        }
        transactions.add(t);
        saveHistory();
    }

    public void removeTransaction(Transaction t) {
        transactions.remove(t);
        saveHistory();
    }

    public List<Transaction> getTransactions() {
        // Encapsulation + data integrity
    }
}
```

Transaction list got updated.

serial	Amount	Category	Date
1	45.0	other	08-05-2025 01:05
Total			45.0

Filter by Amount Filter by Category Clear Filter Undo

UI got updated after we are done with breakpoints

Debugability - Debugger showing the program execution state (usually the variables view) after calling the undo (show the model and/or UI widgets are empty again)

```

RUN AND DEBUG D No Configurations ... TestExample.java 9+ M J ExpenseTrackerModel.java M X J LockSupport.class J ExpenseTra D - C ...
expense_tracker > src > model > J ExpenseTr > add > ↻ iModel > undo()
public class ExpenseTrack > add
}
}

private void saveHistory() {
    while (historyIndex < transactionHistory.size() - 1) {
        transactionHistory.remove(transactionHistory.size() - 1);
    }

    // Save the current history of transactions to the history
    transactionHistory.add(new ArrayList<>(transactions));
    historyIndex++;
}

public void undo() {
    if (historyIndex > 0) { historyIndex = 2
        transactionHistory.remove(historyIndex);
        historyIndex--;
        transactions = new ArrayList<>(transactionHistory.get(historyIndex));
    } else {
        JOptionPane.showMessageDialog(
            parentComponent: null,
            message: "No more actions to perform an undo.",
            title: "Undo Error",
            JOptionPane.ERROR_MESSAGE
        );
    }
}

```

Initial state with 3 transactions before when debugger entered the Undo block.

```

RUN AND DEBUG D No Configurations ... TestExample.java 9+ M J ExpenseTrackerModel.java M X J LockSupport.class J ExpenseTra D - C ...
expense_tracker > src > model > J ExpenseTr > add > ↻ iModel > undo()
public class ExpenseTrack > add
}

private void saveHistory() {
    while (historyIndex < transactionHistory.size() - 1) {
        transactionHistory.remove(transactionHistory.size() - 1);
    }

    // Save the current history of transactions to the history
    transactionHistory.add(new ArrayList<>(transactions));
    historyIndex++;
}

public void undo() {
    if (historyIndex > 0) { historyIndex = 1
        transactionHistory.remove(historyIndex); transactionHistory = ArrayList<>
        historyIndex--; historyIndex = 1
        transactions = new ArrayList<>(transactionHistory.get(historyIndex));
    } else {
        JOptionPane.showMessageDialog(
            parentComponent: null,
            message: "No more actions to perform an undo.",
            title: "Undo Error",
            JOptionPane.ERROR_MESSAGE
        );
    }
}

```

The highlighted part shows that one translation got reduced from the history

```
public class ExpenseTrack {
    ...
    private void saveHistory() {
        while (historyIndex < transactionHistory.size() - 1) {
            transactionHistory.remove(transactionHistory.size() - 1);
        }
        // Save the current history of transactions to the history.
        transactionHistory.add(new ArrayList<>(transactions));
        historyIndex++;
    }

    public void undo() {
        if (historyIndex > 0) { historyIndex = 1;
            transactionHistory.remove(historyIndex); transactionHistory = ArrayList<Transaction>.newArrayList();
            historyIndex--; historyIndex = 1;
            transactions = new ArrayList<>(transactionHistory.get(historyIndex));
        } else {
            JOptionPane.showMessageDialog(
                parentComponent: null,
                message: "No more actions to perform an undo.",
                title: "Undo Error",
                JOptionPane.ERROR_MESSAGE
            );
        }
    }
}
```

The highlighted part shows that one transaction got reduced from the transaction list to

serial	Amount	Category	Date
1	45.0	other	08-05-2025 01:05
2	100.0	food	08-05-2025 01:07
Total			145.0

Amount: 200 Category: food Add Transaction

Filter by Amount Filter by Category Clear Filter Undo

Transaction got removed from the UI too.

