Export to CSV Feature - Design Document

Feature Overview

This is a design document for functionally adding a CSV Export feature to the Expense Tracker application. This function will allow users to export their transaction history information to a CSV file using a filename specified by the user.

Requirements

1. User can specify the output file name
2. Input validation for the file name
3. First line contains column headers
4. Each subsequent line contains one transaction
5. Must follow MVC architecture, UI design laws, OO principles and best practices

Design Approach

1. MVC Architecture Integration

Model Layer (ExpenseTrackerModel.java)

- No changes required to the existing model
- The model already provides `getTransactions()` method to retrieve all transactions
- Transactions are immutable and contain all necessary data (amount, category, timestamp)

View Layer (ExpenseTrackerView.java)

New UI Components:

- Add "Export to CSV" button in the button panel
- Add a text field for filename input with label "Export Filename:"
- Add a file chooser dialog (optional enhancement) using JFileChooser

Layout Changes:

- Add export components to the existing button panel or create a new export panel
- Position near the filter controls for logical grouping of utility features

UI Design Laws Applied:

- **Visibility:** Export button clearly labeled and visible
- **Feedback:** Show success/error messages after export attempt
- **Consistency:** Follow existing button styling and layout patterns
- **Help:** Placeholder text in filename field showing example: "transactions.csv"

New Methods:

```
public JButton getExportButton()
public String getExportFilename()
public void setExportFilename(String filename)
public void showExportSuccessMessage(String filename)
public void showExportErrorMessage(String error)
```

Controller Layer (ExpenseTrackerController.java)

New Method:

```
public boolean exportToCSV(String filename)
```

Responsibilities:

1. Validate the filename using InputValidation class
2. Get transactions from the model
3. Delegate actual file writing to a new CSVExporter utility class
4. Handle exceptions and update the view with appropriate messages
5. Return true if export succeeds, false otherwise

Wire Export Button:

```
view.getExportButton().addActionListener(e -> {
    String filename = view.getExportFilename();
    if (exportToCSV(filename)) {
        view.showExportSuccessMessage(filename);
    } else {
        view.showExportErrorMessage("Export failed. Check filename and try
again.");
    }
});
```

2. Input Validation (InputValidation.java)

New Method:

```
public static boolean isValidFilename(String filename)
```

Validation Rules:

- Filename cannot be null or empty
- Filename cannot contain invalid characters: / \ : * ? " < > |
- Filename should end with ".csv" (add automatically if missing)
- Filename length should be reasonable (1-255 characters)
- No path traversal attempts (e.g., "../" or "..")

Example Implementation Logic:

```
- Check if null or empty/whitespace -> return false
- Check if contains invalid characters -> return false
- Check length (after adding .csv) -> return false if > 255
- Check for path traversal patterns -> return false
- Return true if all checks pass
```

3. CSV Export Utility Class

New Class: CSVExporter.java (in a new "util" or "export" package)

Following **Open-Closed Principle**: This class is designed to be extended for other export formats (JSON, XML) in the future without modifying existing code.

Class Structure:

```
package util;

import model.Transaction;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

public class CSVExporter {

    private static final String CSV_HEADER = "Amount,Category,Date";
    private static final String CSV_DELIMITER = ",";

    public static boolean exportTransactions(List<Transaction> transactions,
String filename) {
        // Implementation here
    }

    private static String escapeCSV(String value) {
        // Handle commas and quotes in data
    }
}
```

Export Logic:

1. Create/open the file using FileWriter
2. Write header line: "Amount,Category,Date"
3. Iterate through transactions and write each as a CSV line
4. Format: `amount,category,timestamp`
5. Handle special characters (commas in data, quotes) using CSV escaping
6. Close file and return success/failure status
7. Catch IOExceptions and return false on error

CSV Escaping Rules:

- If a field contains a comma, wrap it in double quotes
- If a field contains double quotes, escape them by doubling ("")
- Example: `50.00,"Food, Snacks","28-10-2025 14:30"`

4. OO Design Principles Applied

Single Responsibility Principle:

- View: Only handles UI components and user interaction
- Controller: Coordinates between view and model, handles export logic flow
- CSVExporter: Only responsible for CSV file creation
- InputValidation: Only validates inputs

Open-Closed Principle:

- CSVExporter can be extended to an interface (Exporter) later
- Other export formats (JSON, PDF) can implement the same interface
- Existing code won't need modification to add new export types

Dependency Inversion Principle:

- Controller depends on model interface ant not on the concrete implementation
- Future: Create Exporter interface that CSVExporter implements

Encapsulation:

- All file I/O logic is encapsulated in CSVExporter
- View doesn't know about file operations
- Model doesn't know about export operations

5. Best Practices

No Magic Strings:

- Define constants for CSV header, delimiter, file extension

```
private static final String CSV_EXTENSION = ".csv";
private static final String CSV_HEADER = "Amount,Category,Date";
```

```
private static final String CSV_DELIMITER = ",";
private static final String DEFAULT_FILENAME = "transactions.csv";
```

Error Handling:

- Use try-catch blocks for file I/O operations
- Provide meaningful error messages to users
- Log errors for debugging (optional)
- Never expose stack traces to users

Resource Management:

- Use try-with-resources for FileWriter to ensure proper closure

```
try (FileWriter writer = new FileWriter(filename)) {
    // write operations
} catch (IOException e) {
    // handle error
}
```

Input Sanitization:

- Always validate filename before attempting export
- Escape special characters in CSV data
- Prevent path traversal attacks

User Feedback:

- Show success message with filename: "Exported successfully to: transactions.csv"
- Show error message on failure: "Export failed: Invalid filename"
- Clear the filename field after successful export

6. CSV Output Format

Example Output:

```
Amount,Category,Date
50.00,food,28-10-2025 14:30
100.00,travel,28-10-2025 14:35
75.50,bills,28-10-2025 14:40
```

Empty File Handling:

- If no transactions exist, still write the header line
- Output: Just the header with no data rows

7. Testing Considerations

Test Cases to Add:

1. Export with valid filename
2. Export with invalid filename (special characters)
3. Export with empty filename
4. Export with no file extension (should add .csv)
5. Export empty transaction list
6. Export transactions with special characters in category
7. File write permission errors
8. Verify CSV format correctness

Implementation Summary

Files to Modify:

1. ExpenseTrackerView.java - Add export UI components
2. ExpenseTrackerController.java - Add export button handler and exportToCSV method
3. InputValidation.java - Add isValidFilename method

New Files to Create:

1. CSVExporter.java - Utility class for CSV export functionality

Packages:

- util/ (new package for CSVExporter)

Future Enhancements

- Add file chooser dialog for better UX
- Support multiple export formats (JSON, PDF)
- Export filtered transactions only
- Add export date range selection
- Auto-generate filename with timestamp
- Progress indicator for large exports