Undo Functionality Design Plan

Overview

The Undo functionality allows users to reverse the removal of a transaction. When a user removes a transaction, the application stores the removed transaction in temporary storage (for example, using a stack). If the user clicks the **Undo** button, the most recently removed transaction is restored to the list, and the **Total Cost** is updated accordingly.

Changes to the MVC Components

Model

1. **Add a Stack for Removed Transactions:**

• Introduce a Stack<Transaction> in the ExpenseTrackerModel to store recently removed transactions.

• This stack allows the application to restore the most recently removed transaction when the **Undo** button is clicked.

2. **Update removeTransaction Method:**

• Modify the removeTransaction method to push the removed transaction onto the stack.

3. **Add undoRemoveTransaction Method:**

• Implement a method to pop the most recently removed transaction from the stack and add it back to the list of transactions.

Example Code:

```java
// In ExpenseTrackerModel

private Stack<Transaction> removedTransactions = new Stack<>();
```

```java
public void removeTransaction(Transaction t) {
    transactions.remove(t);
    removedTransactions.push(t); // Store the removed transaction
}

public void undoRemoveTransaction() {
    if (!removedTransactions.isEmpty()) {
        Transaction t = removedTransactions.pop();
        transactions.add(t); // Restore the transaction
    }
}
```

View

1. **Add an “Undo” Button:**

• Add a new JButton labeled **Undo** to the UI.

• Place the button in the appropriate panel (for example, the button panel).

2. **Update the Table:**

• Ensure that the table is refreshed when a transaction is restored via the Undo functionality.

Example Code:

```java
// In the view class

private JButton undoBtn = new JButton("Undo");

public JButton getUndoBtn() {
    return undoBtn;
}
```

Controller

1. **Handle Undo Button Clicks:**

• Add an event listener to the **Undo** button.

• When clicked, call the undoRemoveTransaction method in the model and refresh the view.

2. **Update Total Cost:**

• Recalculate and update the **Total Cost** after restoring a transaction.

Example Code:

```
Unset
// In the controller class

view.getUndoBtn().addActionListener(e -> {
    model.undoRemoveTransaction();
    refresh(); // Refresh view to update the table and total cost
});
```

File Structure

• **Model:**

• Add the removedTransactions stack.

• Update the removeTransaction method.

• Add the undoRemoveTransaction method.

• **View:**

• Add an **Undo** button.

• Expose the button through a getter method.

• **Controller:**

• Add an event listener for the **Undo** button.

• Call the model's undoRemoveTransaction method and refresh the view accordingly.

UI Design for Undo Functionality

Undo Button

• **Placement:**

Position the **Undo** button near the transaction list and total cost display.

• **Dynamic State:**

• **Enabled:** When there's an action to undo.

• **Disabled:** When no undoable actions exist (e.g., grayed-out).

Feedback Messages

• **On Remove:** Show a message like **"Transaction removed!"**

• **On Undo:** Show **"Transaction restored!"**

Example Workflow

1. **Removal:**

• User selects a transaction and clicks **Remove**.

• Transaction is deleted, total cost updates, and the **Undo** button becomes enabled.

• Display feedback: **"Transaction removed!"**

2. **Undo:**

• User clicks **Undo**.

• The most recent transaction is restored, the total cost is updated, and feedback **“Transaction restored!”** is shown.

• If no transactions remain in the undo stack, disable the **Undo** button.