# Reflection log

**Class Definition:** The `LocalBankGUI` class is designed to provide a simple graphical user interface (GUI) for managing bank account transactions. It allows users to deposit, withdraw, check their balance, and delete accounts.

**Main Method:** The `main` method is the application's entry point. It utilizes `EventQueue.invokeLater` to ensure the GUI is created on the Event Dispatch Thread, adhering to best practices in Java Swing development. This approach ensures that the GUI remains responsive and thread-safe.

```
public static void main(String[] args) {
    EventQueue.invokeLater(() -> {
        try {
            LocalBankGUI window = new LocalBankGUI();
            window.frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}
```

**Constructor:** The constructor (`LocalBankGUI()`) initializes the GUI by calling the `initialize()` method, setting up the necessary components and layout for user interaction.

```
public LocalBankGUI() {
    initialize();
}
```

**Initialize Method:** The `initialize` method sets up the main window (`JFrame`) and other UI components:

- **JFrame:** Serves as the main window of the application.
- **JPanel:** Not explicitly defined, but the main content pane serves a similar purpose.
- **JLabel:** Used to display prompts and messages to the user.
- **JTextField:** Fields for user input such as account number, amount, and user names.
- **JComboBox:** Provides a dropdown list for users to select the action they want to perform (e.g., deposit, withdraw).

```
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 450, 400);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(null);
```

**Components:**

- **JComboBox (actionComboBox):** Allows users to choose between different banking actions. The selected action determines what happens when the "Process Transaction" button is clicked.
- **JTextField:** Input fields for account number, amount, first name, last name, and beginning balance. These fields facilitate user data entry.
- **JLabel (accountInfoLabel):** Displays feedback and results of transactions, enhancing user interaction and experience.

```
JLabel lblAccountNumber = new JLabel("Account number:");
lblAccountNumber.setBounds(20, 60, 120, 20);
frame.getContentPane().add(lblAccountNumber);


accountNumberField = new JTextField();
accountNumberField.setBounds(160, 60, 180, 20);
frame.getContentPane().add(accountNumberField);


JLabel lblAmount = new JLabel("Amount (if applicable):");
lblAmount.setBounds(20, 90, 180, 20);
frame.getContentPane().add(lblAmount);


amountField = new JTextField();
amountField.setBounds(210, 90, 130, 20);
frame.getContentPane().add(amountField);


JLabel lblFirstName = new JLabel("First Name:");
lblFirstName.setBounds(20, 120, 100, 20);
frame.getContentPane().add(lblFirstName);


firstNameField = new JTextField();
firstNameField.setBounds(160, 120, 180, 20);
frame.getContentPane().add(firstNameField);
```

**Button ActionListener:** An `ActionListener` is attached to the "Process Transaction" button. When clicked, it retrieves the selected action from the combo box and processes the transaction accordingly. The listener leverages a method (`processTransaction`) that handles the logic based on user input.

```
private void processTransaction(String action) {
    String accountNumber = accountNumberField.getText();
    String amount = amountField.getText();
    String firstName = firstNameField.getText();
    String lastName = lastNameField.getText();
    String beginningBalance = beginningBalanceField.getText();
```

**Transactions Logic:** The `processTransaction` method contains the core logic for handling banking operations:

- **Deposit:** Adds funds to the account and updates the balance.
- **Withdraw:** Deducts funds, ensuring sufficient balance is available.
- **Check Balance:** Displays the current balance without altering it.
- **Delete Account:** Resets account information and displays a deletion message if the account exists.

```
// Dropdown for selecting action
JComboBox<String> actionComboBox = new JComboBox<>();
actionComboBox.addItem("Deposit");
actionComboBox.addItem("Withdraw");
actionComboBox.addItem("Check Balance");
actionComboBox.addItem("Delete Account");
actionComboBox.setBounds(160, 20, 120, 20);
frame.getContentPane().add(actionComboBox);
```

**Error Handling:** The application includes basic error handling, such as checking for insufficient funds during withdrawals and confirming the existence of accounts before deletion. This enhances the robustness of the GUI.

```
private void deleteAccount(String accountNumber) {
    if (accountExists) {
        balance = 0;
        accountExists = false;
        accountInfoLabel.setText("Account " + accountNumber + " deleted.");
        clearFields();
    } else {
        accountInfoLabel.setText("Account does not exist.");
    }
}
```

**User Feedback:** The `accountInfoLabel` provides real-time feedback on actions taken, helping users understand the outcomes of their transactions.

```
if (action.equals("Deposit")) {
    double transactionAmount = Double.parseDouble(amount);
    balance += transactionAmount;
    accountInfoLabel.setText("Deposited: $" + transactionAmount);
} else if (action.equals("Withdraw")) {
    double transactionAmount = Double.parseDouble(amount);
    if (transactionAmount <= balance) {
        balance -= transactionAmount;
        accountInfoLabel.setText("Withdrew: $" + transactionAmount);
    } else {
        accountInfoLabel.setText("Insufficient funds for withdrawal.");
    }
} else if (action.equals("Check Balance")) {
    accountInfoLabel.setText(String.format("Current Balance: $%.2f", balance));
    return; // No need to update further info
}
```