```c
#include "Bank.h"
#include "Colors.h"
#include "TerminalUI.h"
#include <stdio.h>
#include <string.h>

// Transaction counters for summary
int deposit_count = 0;
int withdraw_count = 0;
int transfer_count = 0;
int failed_withdraw_count = 0;
int failed_transfer_count = 0;

void Bank_init(Bank *bank, int numAccounts, double initialBalance)
{
    bank->numAccounts = numAccounts;
    for (int i = 0; i < numAccounts; ++i)
    {
        Account_init(&bank->accounts[i], i + 1, initialBalance);
    }

    // Display pretty initialization banner
    printHeader("BANK SYSTEM INITIALIZED");

    // Print account initialization table
    int ids[MAX_ACCOUNTS];
    double balances[MAX_ACCOUNTS];

    for (int i = 0; i < numAccounts; i++)
    {
        ids[i] = i + 1;
        balances[i] = initialBalance;
    }

    printAccountTable(numAccounts, ids, balances);
}

void Bank_processTransaction(Bank *bank, Transaction txn)
{
    Account *from = Bank_findAccount(bank, txn.fromAccountId);
    if (!from)
    {
        printAccountNotFound(txn.fromAccountId);
        return;
    }

    if (txn.type == DEPOSIT)
    {
        Account_deposit(from, txn.amount);
```

```
50          printTransactionTable(DEPOSIT, txn.amount, txn.fromAccountId, -1,
    Account_getBalance(from), 0);
51          deposit_count++;
52      }
53      else if (txn.type == WITHDRAW)
54      {
55          int success = Account_withdraw(from, txn.amount);
56          if (success)
57          {
58              printTransactionTable(WITHDRAW, txn.amount, txn.fromAccountId, -1,
    Account_getBalance(from), 0);
59              withdraw_count++;
60          }
61          else
62          {
63              printFailedTransaction(WITHDRAW, txn.fromAccountId,
    Account_getBalance(from), txn.amount);
64              failed_withdraw_count++;
65          }
66      }
67      else if (txn.type == TRANSFER)
68      {
69          Account *to = Bank_findAccount(bank, txn.toAccountId);
70          if (to)
71          {
72              int success = Account_transfer(from, to, txn.amount);
73              if (success)
74              {
75                  printTransactionTable(TRANSFER, txn.amount, txn.fromAccountId,
    txn.toAccountId,
76                                        Account_getBalance(from),
    Account_getBalance(to));
77                  transfer_count++;
78              }
79              else
80              {
81                  printFailedTransaction(TRANSFER, txn.fromAccountId,
    Account_getBalance(from), txn.amount);
82                  failed_transfer_count++;
83              }
84          }
85          else
86          {
87              printDestinationNotFound(txn.toAccountId);
88          }
89      }
90  }
91
92  void Bank_printSummary()
93  {
94      printTransactionSummary(deposit_count, withdraw_count, transfer_count,
```

```
                              failed_withdraw_count, failed_transfer_count);
}

Account *Bank_findAccount(Bank *bank, int id)
{
    for (int i = 0; i < bank->numAccounts; ++i)
    {
        if (Account_getId(&bank->accounts[i]) == id)
            return &bank->accounts[i];
    }
    return NULL;
}

void Bank_displayBalances(Bank *bank)
{
    int ids[MAX_ACCOUNTS];
    double balances[MAX_ACCOUNTS];

    for (int i = 0; i < bank->numAccounts; i++)
    {
        ids[i] = Account_getId(&bank->accounts[i]);
        balances[i] = Account_getBalance(&bank->accounts[i]);
    }

    printHeader("CURRENT ACCOUNT BALANCES");
    printAccountTable(bank->numAccounts, ids, balances);
}
```