```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <unistd.h>
#include "Bank.h"
#include "Transaction.h"
#include "Colors.h"
#include "TerminalUI.h"

#define MAX_TRANSACTIONS 1000

Bank bank; // Global bank object

void *transactionThread(void *arg)
{
    Transaction *txn = (Transaction *)arg;

    // Random delay to simulate real-life transaction time
    int delayMs = rand() % 1001; // 0 to 1000 milliseconds
    usleep(delayMs * 1000);      // sleep in microseconds

    Bank_processTransaction(&bank, *txn);

    pthread_exit(NULL);
}

int main()
{
    srand(time(NULL));

    // Welcome screen
    system("clear");

    int numTransactions;
    printf("%s➤ Enter number of simultaneous transactions to simulate: %s", BOLD_CYAN, RESET);
    scanf("%d", &numTransactions);
    printf("\n");

    if (numTransactions > MAX_TRANSACTIONS)
    {
        printf("%s", BOLD_RED);

        printf("╔═══════════════════════════════════════════════════════════════════════════╗\n");
        printf("║ Error: Maximum allowed transactions is %-29d ║\n", MAX_TRANSACTIONS);

        printf("╚═══════════════════════════════════════════════════════════════════════════╝\n");
        printf("%s", RESET);
        return 1;
    }

    pthread_t threads[MAX_TRANSACTIONS];
    Transaction *transactions[MAX_TRANSACTIONS];

    Bank_init(&bank, 5, 1000.0); // 5 accounts starting with $1000

    system("clear");

    printHeader("TRANSACTION SIMULATION STARTED");
    printf("%s%sSimulating %d concurrent transactions...%s\n\n", BOLD_YELLOW, BOX_V, numTransactions, RESET);
```

```
59
60      for (int i = 0; i < numTransactions; ++i)
61      {
62          int action = rand() % 3 + 1;
63          int fromAccount = (rand() % 5) + 1;
64          int toAccount = (rand() % 5) + 1;
65          while (toAccount == fromAccount)
66          {
67              toAccount = (rand() % 5) + 1;
68          }
69          double amount = (rand() % 500) + 1; // $1 - $500
70
71          transactions[i] = (Transaction *)malloc(sizeof(Transaction));
72          if (!transactions[i])
73          {
74              perror("malloc failed");
75              return 1;
76          }
77
78          if (action == 1)
79          { // Deposit
80              Transaction_init(transactions[i], DEPOSIT, fromAccount, amount, -1);
81          }
82          else if (action == 2)
83          { // Withdraw
84              Transaction_init(transactions[i], WITHDRAW, fromAccount, amount, -1);
85          }
86          else
87          { // Transfer
88              Transaction_init(transactions[i], TRANSFER, fromAccount, amount,
    toAccount);
89          }
90
91          pthread_create(&threads[i], NULL, transactionThread, transactions[i]);
92      }
93
94      for (int i = 0; i < numTransactions; ++i)
95      {
96          pthread_join(threads[i], NULL);
97          free(transactions[i]);
98      }
99
100     // Print transaction summary and final balances
101     Bank_printSummary();
102     Bank_displayBalances(&bank);
103
104     printCompletionMessage();
105     printf("\n");
106
107     return 0;
108 }
```