```
#include <iostream>
#include <stdexcept>
#include <string>
using namespace std;

// Custom exception for negative deposit amounts
class NegativeDepositException : public runtime_error {
public:
    NegativeDepositException() : runtime_error("Deposit amount cannot be negative.") {}
};

// Custom exception for insufficient funds on withdrawal
class InsufficientFundsException : public runtime_error {
public:
    InsufficientFundsException() : runtime_error("Insufficient funds for withdrawal.") {}
};

// BankAccount class to manage individual account details and operations
class BankAccount {
private:
    int accountNumber;
    string accountHolderName;
    double balance;

public:
    // Constructor to initialize account details
    BankAccount(int accNumber, const string& accHolderName, double initialBalance)
        : accountNumber(accNumber), accountHolderName(accHolderName),
balance(initialBalance) {
        if (initialBalance < 0) throw runtime_error("Initial balance cannot be negative.");
    }
```

```
    // Deposit function to add money to the account
    void deposit(double amount) {
        if (amount < 0) throw NegativeDepositException();
        balance += amount;
        cout << "Deposit successful.\n";
    }

    // Withdraw function to subtract money from the account
    void withdraw(double amount) {
        if (amount > balance) throw InsufficientFundsException();
        balance -= amount;
        cout << "Withdrawal successful.\n";
    }

    // Display the current balance and account information
    void displayBalance() const {
        cout << "Account Number: " << accountNumber << "\n"
             << "Holder Name: " << accountHolderName << "\n"
             << "Current Balance: KES" << balance << "\n";
    }

    // Get the account number
    int getAccountNumber() const {
        return accountNumber;
    }
};

// Find an account by its number
BankAccount* findAccount(BankAccount* accounts[], int accountCount, int accNum) {
    for (int i = 0; i < accountCount; ++i) {
```

```
        if (accounts[i]->getAccountNumber() == accNum)
            return accounts[i];
    }
    return nullptr; // Return nullptr if account not found
}

// Function to display all accounts
void displayAllAccounts(BankAccount* accounts[], int accountCount) {
    if (accountCount == 0) {
        cout << "No accounts.\n";
        return;
    }

    cout << "\nAll Bank Accounts:\n";
    for (int i = 0; i < accountCount; ++i) {
        cout << "----------------------\n";
        accounts[i]->displayBalance();
        cout << "----------------------\n";
    }
}

int main() {
    const int maxAccounts = 5;
    BankAccount* accounts[maxAccounts];
    int accountCount = 0;

    int choice;
    while (true) {
        // Display menu options
        cout<< "\nBank Management System\n";
        cout << "1. Add New Account\n";
```

```
cout << "2. Deposit\n";
cout << "3. Withdraw\n";
cout << "4. Display Balance\n";
cout << "5. Display All Accounts\n";
cout << "6. Exit\n";
cout << "Choose an option: ";
cin >> choice;

if (choice == 6) break; // Exit the program

int accNum;
double amount;
BankAccount* account = nullptr;

switch (choice) {
   case 1:
      if (accountCount < maxAccounts) {
         string accName;
         double initialBalance;

         // Gather account creation details
         cout << "Enter account number: ";
         cin >> accNum;
         cout << "Enter account holder name: ";
         cin.ignore();
         getline(cin, accName);
         cout << "Enter initial balance: ";
         cin >> initialBalance;

         try {
            // Create and add a new account
```

```
                accounts[accountCount++] = new BankAccount(accNum, accName,
initialBalance);
                cout << "Account created successfully.\n";
            } catch (const runtime_error& e) {
                cout << "Error: " << e.what() << "\n";
            }
        } else {
            cout << "Account limit reached.\n";
        }
        break;

    case 2:
        // Deposit operation
        cout << "Enter account number for deposit: ";
        cin >> accNum;
        account = findAccount(accounts, accountCount, accNum);

        if (account) {
            cout << "Enter deposit amount: ";
            cin >> amount;
            try {
                account->deposit(amount);
            } catch (const NegativeDepositException& e) {
                cout << "Error: " << e.what() << "\n";
            }
        } else {
            cout << "Account not found.\n";
        }
        break;

    case 3:
```

```
    // Withdraw operation
    cout << "Enter account number for withdrawal: ";
    cin >> accNum;
    account = findAccount(accounts, accountCount, accNum);

    if (account) {
        cout << "Enter withdrawal amount: ";
        cin >> amount;
        try {
            account->withdraw(amount);
        } catch (const InsufficientFundsException& e) {
            cout << "Error: " << e.what() << "\n";
        }
    } else {
        cout << "Account not found.\n";
    }
    break;

case 4:
    // Display balance operation
    cout << "Enter account number to display balance: ";
    cin >> accNum;
    account = findAccount(accounts, accountCount, accNum);

    if (account) {
        account->displayBalance();
    } else {
        cout << "Account not found.\n";
    }
    break;
```

```
            case 5:
                // Display all accounts
                displayAllAccounts(accounts, accountCount);
                break;



            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    // Clean up dynamically allocated memory
    for (int i = 0; i < accountCount; ++i) {
        delete accounts[i];
    }

    return 0;
}
```