Olympic Bank
P0 Banking App

Tech Stack:
C# on .NET Framework 4.7
SQL Server

Jamal Reynolds, Jr. Developer
Github @daemonmal

REVATURE

Overview

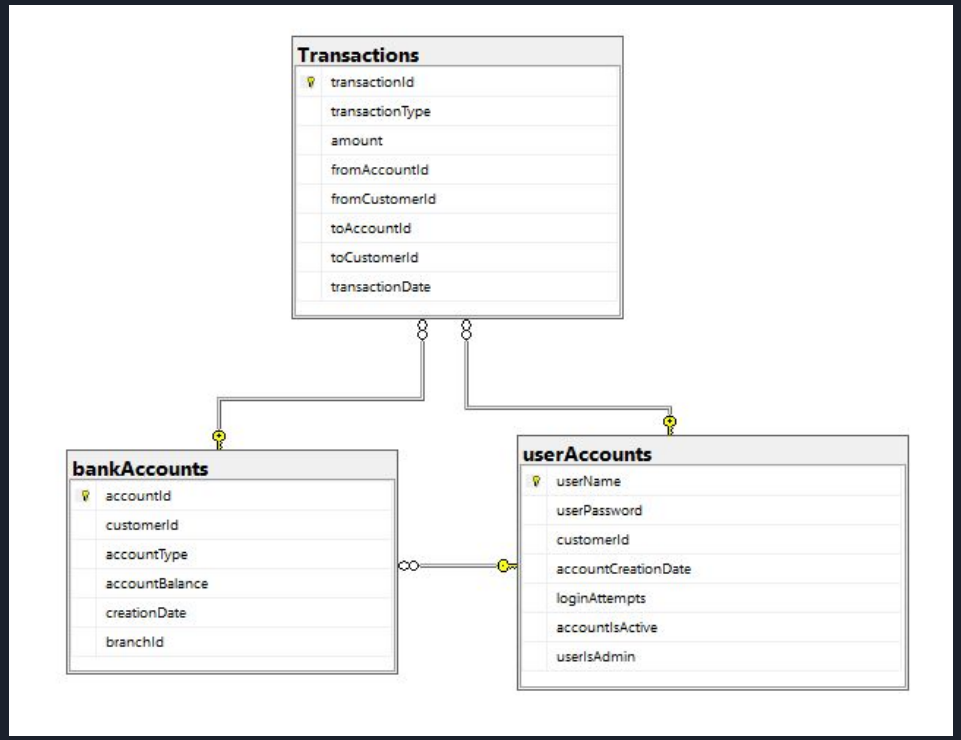
- Introduction
- ER Diagram
- OOPS
- SOLID Design
- Future improvements

Introduction

Project Goals:

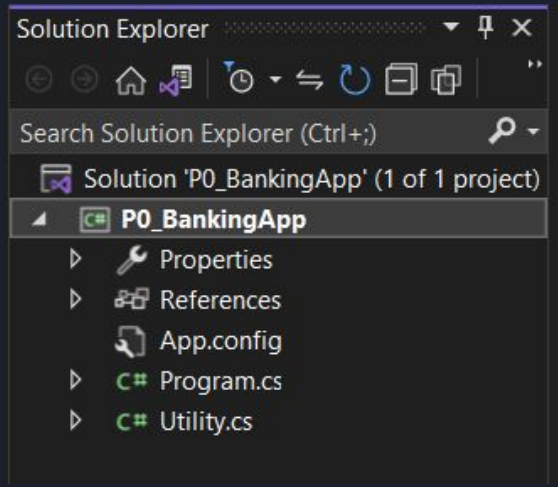
- Develop a console application Banking app with separate admin/employee and customer menu functionality
- Utilize 3NF SQL Server Database, ADO.NET, and libraries to implement 2-Tier architecture
- Implement exception handling and logging using Serilog
- Utilize best practices and design principles

ER Diagram


Transactions
transactionId
transactionType
amount
fromAccountId
fromCustomerId
toAccountId
toCustomerId
transactionDate
bankAccounts
accountId
customerId
accountType
accountBalance
creationDate
branchId
userAccounts
userName
userPassword
customerId
accountCreationDate
loginAttempts
accountIsActive
userIsAdmin

OOPS - Abstraction

- Implementation is hidden from the user through the use of a library and helper methods




```
#region Customer Menu
//customer menu
Console.WriteLine("Weclome to your Customer Account. What would you like to do today?");
Console.WriteLine("1. Check Account Balances");
Console.WriteLine("2. Withdraw Funds");
Console.WriteLine("3. Deposit Funds");
Console.WriteLine("4. Transfer Funds");
Console.WriteLine("5. View Last 10 Transactions");
Console.WriteLine("6. Change Account Password");
Console.WriteLine("7. Logout");
Console.Write("Menu Selection: ");
int selectionCus = Convert.ToInt32(Console.ReadLine());

bool loggedInAsCustomer = true;

while (loggedInAsCustomer)
{

    #region Menu Switches
    switch (selectionCus)
    {

        #region Case 1: Check Account Balances
        case 1:
            Console.Clear();
            Console.WriteLine("1. Check All Account Balances");
            Console.WriteLine("Your open accounts:");

            // list available bank accounts in table format
            _ = new StringBuilder();
            StringBuilder sb2 = utilityObj.GenerateBankAccountTable(v_customerId);
            Console.WriteLine(sb2);

            Console.ReadLine();
            Console.Clear();
            break;
        #endregion
```

OOPS - Encapsulation

- Define properties for classes
- Use setter and getter methods to provide control over data
- Use access modifiers as wrappers

```
0 references
public abstract class GetBankAccountBase
{

    0 references
    public abstract BankAccount GetAccountDetails(int p_customerId);
}
```

```
namespace P0_BankingAppDAL_LIB.Accounts
{
    2 references
    public enum AccountType
    {
        Checking,
        Savings,
        Loan
    }
    13 references
    public class BankAccount
    {
        2 references
        public int accountId { get; set; }
        2 references
        public int customerId { get; set; }
        2 references
        public AccountType accountType { get; set; }
        2 references
        public double accountBalance { get; set; }
        1 reference
        public DateTime accountCreationDate { get; set; }
        2 references
        public int branchId { get; set; }

    }
}
```

OOPS - Inheritance

- Base classes were used throughout to allow subclass inheritance of methods, properties, and fields

```
namespace P0_BankingAppDAL_LIB.Login.Users.Get
{
    0 references
    public class GetCustomerIdSub : GetCustomerIdBase
    {
        SqlConnection connect = new SqlConnection("server=localhost\\TRAINING;database=userAccountDB; Integrated Security=True");

        1 reference
        public override int GetCustomerId(string p_userName)
        {
            int customerId;
            SqlCommand cmdGetId = new SqlCommand("select customerId from userAccounts where userName=@uName", connect);
            cmdGetId.Parameters.AddWithValue("@uName", p_userName);
            connect.Open();
            customerId = Convert.ToInt32(cmdGetId.ExecuteScalar());

            connect.Close();
            return customerId;
        }
        1 reference
        public override int GetCustomerId2(int p_accountId)
        {
            int customerId;
            SqlCommand cmdGetId = new SqlCommand("select customerId from userAccounts where accountId=@accId", connect);
            cmdGetId.Parameters.AddWithValue("@accId", p_accountId);

            connect.Open();
            customerId = Convert.ToInt32(cmdGetId.ExecuteScalar());
            connect.Close();
            return customerId;
        }
    }
}
```

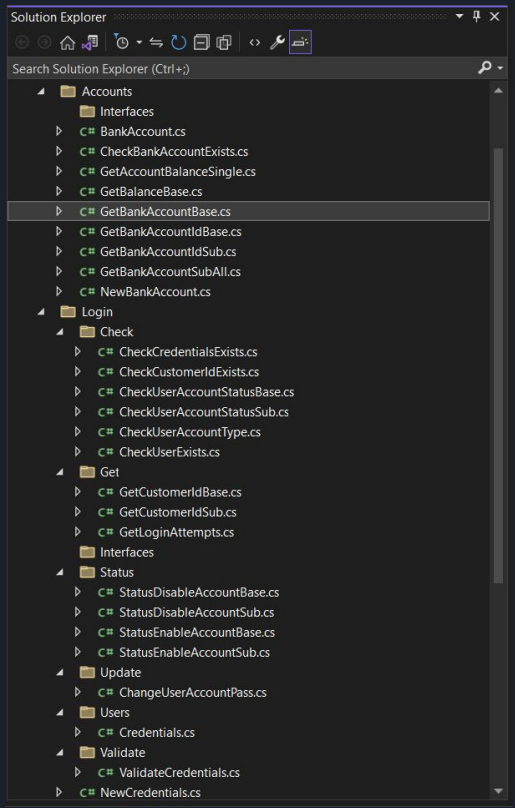
OOPS - Polymorphism

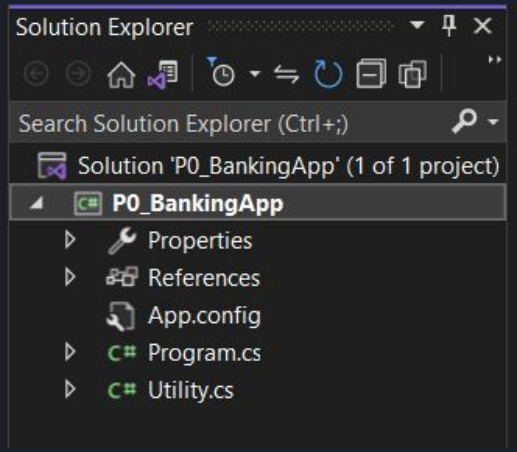
- Methods with no implementation in base classes
- Polymorphism through method overriding
- Increase code flexibility and scalability

```
namespace P0_BankingAppDAL_LIB.Login.Users.Get
{
    0 references
    public class GetCustomerIdSub : GetCustomerIdBase
    {
        SqlConnection connect = new SqlConnection("server=localhost\\TRAINING;database=userAccountDB; Integrated Security=True");

        1 reference
        public override int GetCustomerId(string p_userName)
        {
            int customerId;
            SqlCommand cmdGetId = new SqlCommand("select customerId from userAccounts where userName=@uName", connect);
            cmdGetId.Parameters.AddWithValue("@uName", p_userName);
            connect.Open();
            customerId = Convert.ToInt32(cmdGetId.ExecuteScalar());

            connect.Close();
            return customerId;
        }
        1 reference
        public override int GetCustomerId2(int p_accountId)
        {
            int customerId;
            SqlCommand cmdGetId = new SqlCommand("select customerId from userAccounts where accountId=@accId", connect);
            cmdGetId.Parameters.AddWithValue("@accId", p_accountId);

            connect.Open();
            customerId = Convert.ToInt32(cmdGetId.ExecuteScalar());
            connect.Close();
            return customerId;
        }
    }
}
```

SOLID - Single Responsibility Principle

- A class should have only one responsibility
- Use of abstraction
- Base classes and subclasses





Future Improvements

- Implement logging with Serilog connected to a separate database
- Implement ATM Card feature
- Implement notification and message system into customer accounts
- Add feature for employees to add a new bank account for existing customers
- Add more helper methods to reduce complexity of Main method in Program.cs
  - Validation
  - Menu items
  - etc.
- Improve UI through code-generated menu design
- Convert user password input to asterisks in console app
- Security features such as encryption and sanitation of user input
- Perform Unit Testing
- Provide documentation (test cases)
- Use design patterns and SOLID more extensively to improve code efficiency
- Improve and refine exception handling techniques

Demo

Questions?