Final C# Project:
ATM/Banking Application

ISDS 309 S03
Spring 2024
Professor Hoda Diba

Prepared By: ISDS Group #5

Moises Gonzalez
Blake Demarest
Kylie Ebrahimi
Jesus Gutierrez
Sean Smith

Project Summary

The Simple ATM Interface project aims to create a user-friendly interface for conducting basic ATM transactions. It will include a login screen for user authentication and a transaction screen for performing common ATM operations. Our project focuses on a few fundamental components, including user authentication and transaction execution. The login screen serves as the gateway to the banking system, ensuring that only authorized users can access their accounts. Once authenticated, users are granted access to a transaction screen. From checking their account balance to depositing funds, withdrawing, and transferring money, our interface offers services tailored to meet the needs of our banking customers.

Project Detail Page

The Simple ATM Interface program offers users a seamless and intuitive banking experience through an interactive Windows Forms interface. With a focus on user convenience and security, each customer is assigned a unique username and password, securely stored in a designated file. Upon successful authentication, users gain access to a comprehensive range of banking functionalities, including checking their account balance, depositing funds, withdrawing cash, and transferring funds between their accounts.

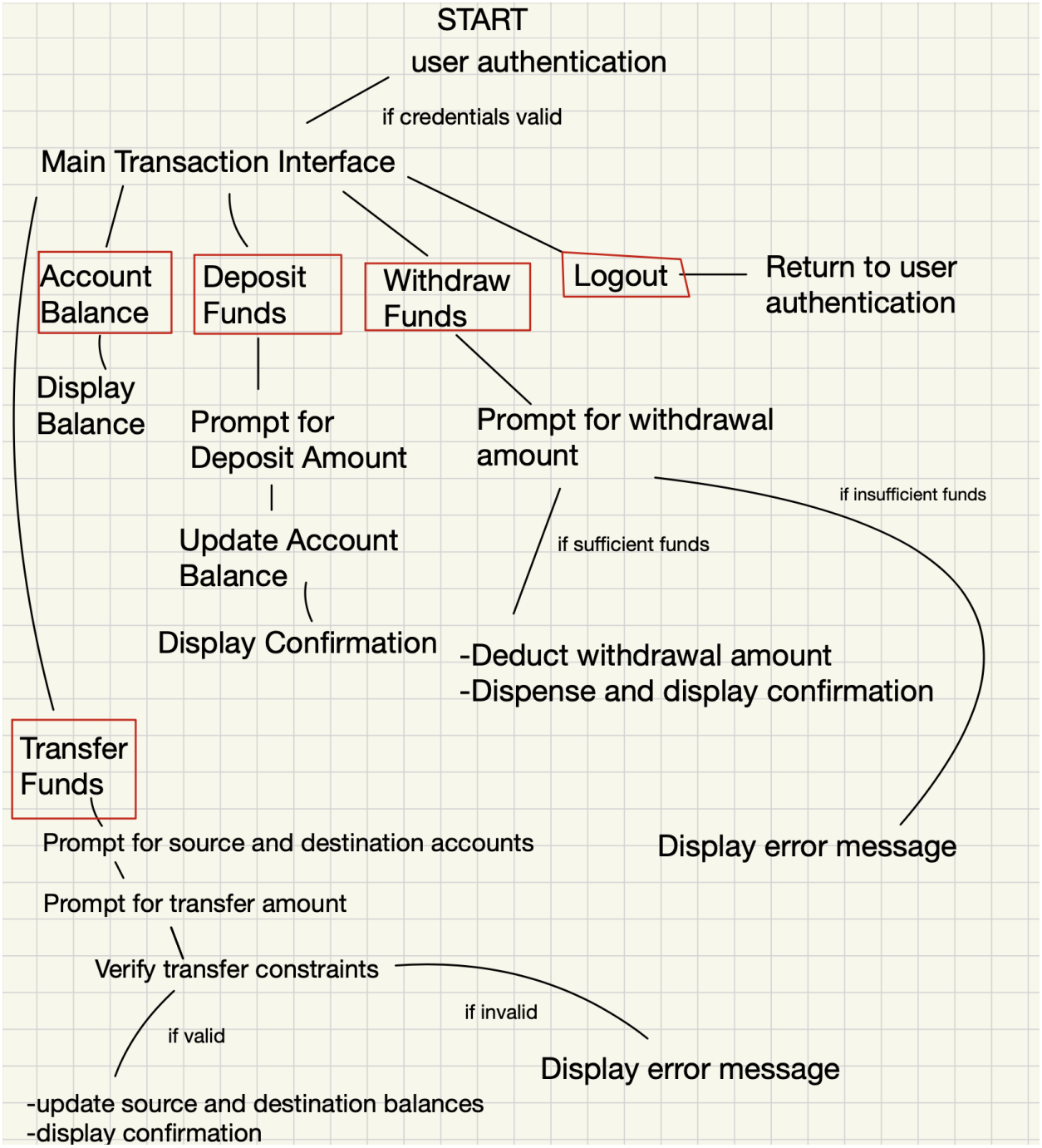
Behind the scenes, the program utilizes advanced data management techniques to ensure efficient and secure access to user information. By establishing a stream to the specified file, the program effectively organizes user data into arrays, enabling swift retrieval and manipulation of relevant information tied to each individual's username. This streamlined approach not only enhances user experience but also reinforces data privacy and security measures.

The program's transactional capabilities are straightforward and user-friendly. Deposits increase the user's available balance by adding the deposited amount to the relevant array and updating the file accordingly. Conversely, withdrawals deduct funds from the user's balance, ensuring accurate and real-time account management. Transfers between accounts are seamlessly executed, with the program manipulating both source and destination arrays to reflect the transaction accurately in the file.

Users receive a comprehensive banking receipt generated in a log file format after completing their transactions. This receipt includes essential details such as the user's username, the timestamp of their platform access, and a record of the tasks they completed during their session. By logging this information and generating receipts, the program enhances transparency and accountability, providing users with a tangible record of their banking activities.

In summary, the Simple ATM Interface program represents a modern approach to banking, combining user-centric design principles with data management techniques. By prioritizing simplicity, security, and accountability, the program empowers users to manage their finances confidently and conveniently, ushering in a new era of digital banking excellence.

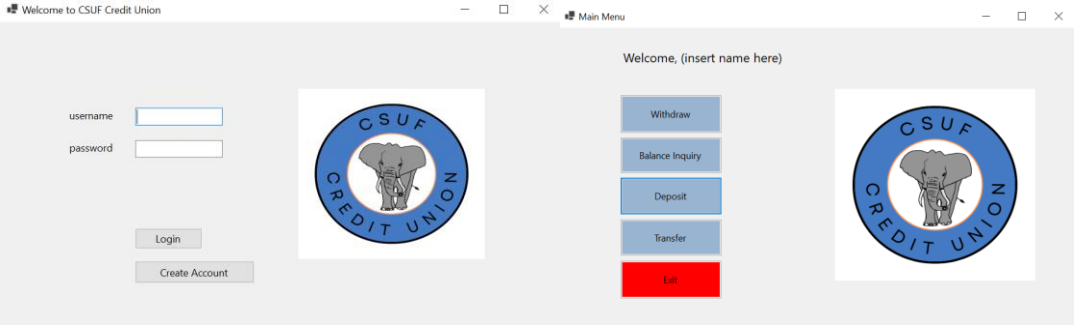
Logic Flow Chart



User Guide Pages

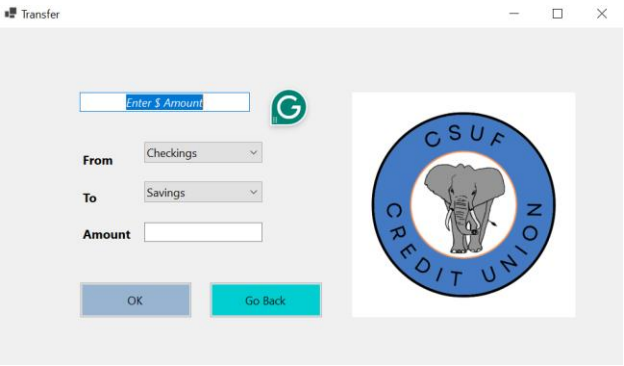
1. Logging In

-The program prompts users to enter both their username and password on the login screen.
-User input is validated against the stored usernames and passwords in a designated text file.
-If the provided credentials match any existing records, the program retrieves the user's information using the GetUserInformation command.
-The program then proceeds to utilize only the attributes associated with the authenticated user throughout the session, ensuring privacy and security.



2. Transfers

-Users initiate transfers by selecting the appropriate option from the transaction menu.
-The transfer function employs nested if-else statements to validate user inputs and manage error control.
-Parameters such as transferring to the same account or exceeding available funds are checked.
-If the transfer meets the correct parameters, the program updates the available balances in both accounts and displays a success message along with the new remaining balance.



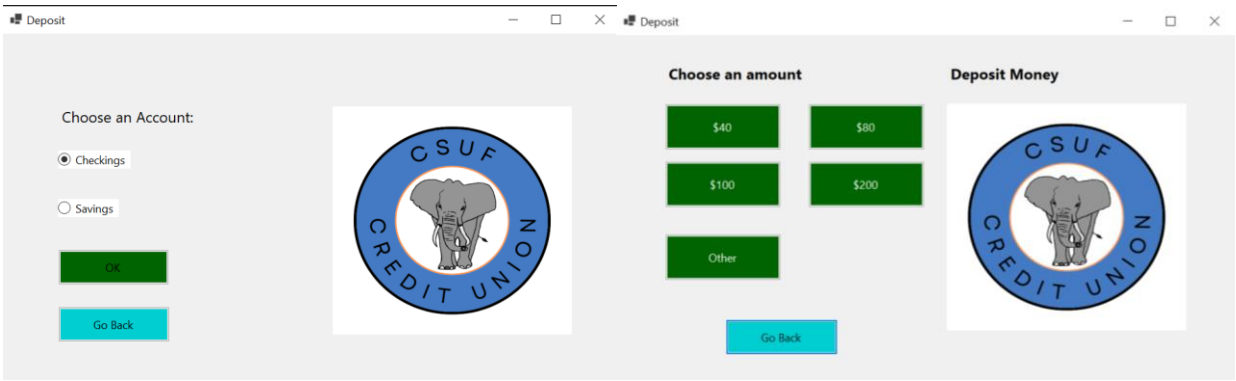
3. Deposits

-Users initiate deposits by selecting the deposit option from the transaction menu.
-The deposit function writes to the master account file through a stream, updating the specific user's account balance.

-Users can enter any deposit amount, and upon completion, the program alerts the user of the successful transaction.

4. Withdrawals

-Similar to deposits, users initiate withdrawals by selecting the withdrawal option from the transaction menu.
-The withdrawal function reverses the process of deposits, updating the user's account balance accordingly.
-Error control mechanisms ensure that users can only withdraw funds available in their account.
-If users attempt invalid actions, such as withdrawing more than their available balance, an error message prompts them to enter a valid value.



5. Logout/Receipts

-The program updates the log file with the transaction details when users log out of the ATM.
-A personalized receipt is generated using stream writer commands, providing users with a record of their account changes.
-The receipt includes a before and after section, displaying the changes in account balances after each transaction.

```
Session Start: 5/9/2024 11:18:31 AM
User: blake d
Checking account balance: $400
Savings account balance: $1300
====================
Transaction Type: Withdrawal
Amount: $200
Account Type: Checking
Balance After Transaction: $200
Timestamp: 5/9/2024 11:18:36 AM
--------------------
Session End: 5/9/2024 11:18:38 AM
User: blake d
Checking account balance: $200
Savings account balance: $1300
====================
```

Sample receipt after withdrawing $200 From checkings

Features

Login Screen: Users input their credentials (username and password) to access ATM functionalities.

Transaction Screen: After login, users are presented with a menu of available transactions, including withdrawal, deposit, and balance inquiry.

Receipt Output: A receipt is printed at the end of each transaction, providing detailed information such as date, time, and transaction details.

Transaction Storage: Transactions are stored until the end of the day, ensuring accurate record-keeping.

Efficient Data Management: Emphasis is placed on efficient storage and retrieval of information to optimize program performance.

Error Handling: Comprehensive error handling is implemented to manage unexpected scenarios, providing informative messages to guide users.

Windows Forms (WinForms): The user interface is developed using WinForms, offering a familiar and intuitive environment for users.

Code Print-Out Pages

-Main Form (Form1)
-Form screenshots
-Content of Form1.cs (and the ".cs" of any other forms you may have, if you have multiple forms)
-Same for any additional Form you may use

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.Data;
 5  using System.Drawing;
 6  using System.Linq;
 7  using System.Text;
 8  using System.Threading.Tasks;
 9  using System.Windows.Forms;
10  using static ISDS309FinalProject.Form1_LoginFrm;
11
12  namespace ISDS309FinalProject
13  {
14      public partial class mainmenuFrm : Form
15      {
16          public bool IsWithdrawal { get; set; } // True for withdrawal,
              false for deposit
17
18          public mainmenuFrm()
19          {
20              InitializeComponent();
21              this.FormClosing += FormUtilities.CloseFormHandler; //Calling
                  the close handler.
22          }
23
24
25
26          private void ShowForm(Form form)
27          {
28              this.Hide(); // Hide the main menu
29
30              form.Show(); // Show the form
31          }
32
33          private void pictureBox1_Click(object sender, EventArgs e)
34          {
35
36          }
37
38
39          private void withdrawalBtn_Click(object sender, EventArgs e)
40          {
41              IsWithdrawal = true;
42              ChooseAccountFrm settingsForm = new ChooseAccountFrm(true);
43              FormUtilities.ShowForm(this, settingsForm);
44          }
45
46          private void balanceBtn_Click(object sender, EventArgs e)
47          {
```

Main Menu

```
...ject-master\ISDS309FinalProject-master\MainMenuFrm.cs                    2
48              balanceFrm settingsForm = new balanceFrm();
49              FormUtilities.ShowForm(this, settingsForm);
50          }
51
52          private void depositBtn_Click(object sender, EventArgs e)
53          {
54              IsWithdrawal = false;
55              ChooseAccountFrm settingsForm = new ChooseAccountFrm(false);
56              FormUtilities.ShowForm(this, settingsForm); // Pass false for
                  deposits
57          }
58
59          private void transferBtn_Click(object sender, EventArgs e)
60          {
61              transferFrm settingsForm = new transferFrm();
62              FormUtilities.ShowForm(this, settingsForm);
63          }
64
65          private void MainMenuFrm_Load(object sender, EventArgs e)
66          {
67
68          }
69
70          private void mainmenuFrm_Load_1(object sender, EventArgs e)
71          {
72
73          }
74
75          private void cancelBtn_Click(object sender, EventArgs e)
76          {
77              FormUtilities.CloseApplication(sender, e);
78          }
79      }
80  }
81
```

Form1_LoginFrm

...t-master\ISDS309FinalProject-master\Form1_LoginFrm.cs 1

```
1  using System.Text;
2
3  namespace ISDS309FinalProject
4  {
5      public partial class Form1_LoginFrm : Form
6      {
7          //This handles the application closing
8          public static class FormUtilities
9          {
10             public static void CloseFormHandler(object sender,
                 FormClosingEventArgs e)
11             {
12                 Application.Exit(); // This will terminate the application
                      when user presses the X
13             }
14
15             public static void CloseApplication(object sender, EventArgs
                 e)
16             {
17                 DialogResult result = MessageBox.Show($"Are you sure you
                     want to exit?", "Confirm Exit", MessageBoxButtons.YesNo,
                      MessageBoxIcon.Question);
18                 if (result == DialogResult.Yes)
19                 {
20
21                     Application.Exit(); //Terminates application when a
                      button is pressed
22                 }
23
24             }
25
26             public static void ShowForm(Form currentForm, Form newForm)
27             {
28                 currentForm.Hide();
29                 newForm.FormClosed += (s, args) =>
30                 {
31                     currentForm.Show();
32                 };
33                 newForm.Show();
34             }
35         }
36
37
38
39         public class TransactionLogger
40         {
41             private static string receiptPath;
42
43             // Initialize when user logs in
```

```
44              public static void InitializeReceipt()
45              {
46                  receiptPath = Path.Combine(Application.StartupPath,
                      $"Receipt_{UserSession.CurrentUsername}_
                      {DateTime.Now:yyyy--MM--dd--HH--mmss}.txt");
47                  using (StreamWriter writer = new StreamWriter(receiptPath,
                       true))
48                  {
49                      writer.WriteLine("Session Start: " +
                       DateTime.Now.ToString());
50                      writer.WriteLine("User: " +
                       UserSession.CurrentFirstName + " " +
                       UserSession.CurrentLastName);
51                      writer.WriteLine("Checking account balance: $" +
                       UserSession.CurrentCheckingBalance);
52                      writer.WriteLine("Savings account balance: $" +
                       UserSession.CurrentSavingsBalance);
53                      writer.WriteLine("====================");
54                  }
55              }
56
57              // Call this method to log a transaction
58              public static void LogTransaction(string transactionType,
                  double amount, string accountType, double balanceAfter)
59              {
60                  using (StreamWriter writer = new StreamWriter(receiptPath,
                       true))
61                  {
62                      writer.WriteLine($"Transaction Type:
                       {transactionType}");
63                      writer.WriteLine($"Amount: ${amount}");
64                      writer.WriteLine($"Account Type: {accountType}");
65                      writer.WriteLine($"Balance After Transaction:
                       ${balanceAfter}");
66                      writer.WriteLine($"Timestamp: {DateTime.Now}");
67                      writer.WriteLine("--------------------");
68                  }
69              }
70
71              // Call this at application exit or logout
72              public static void FinalizeReceipt()
73              {
74                  using (StreamWriter writer = new StreamWriter(receiptPath,
                       true))
75                  {
76                      writer.WriteLine("Session End: " +
                       DateTime.Now.ToString());
77                      writer.WriteLine("User: " +
                       UserSession.CurrentFirstName + " " +
```

```
                    UserSession.CurrentLastName);
78                   writer.WriteLine("Checking account balance: $" +
                    UserSession.CurrentCheckingBalance);
79                   writer.WriteLine("Savings account balance: $" +
                    UserSession.CurrentSavingsBalance);
80                   writer.WriteLine("====================");
81                }
82            }
83        }
84
85
86        public class UserDataTracking
87        {
88            //do compsci majors have to do this all the time?? this is
                insane
89            public static string[,] LoadUserData(string filePath)
90            {
91                string[] lines = File.ReadAllLines(filePath);
92                string[,] userData = new string[lines.Length, 7];
93
94                for (int i = 0; i < lines.Length; i++)
95                {
96                    string[] data = lines[i].Split(',');
97                    for (int j = 0; j < data.Length; j++)
98                    {
99                        userData[i, j] = data[j];
100                   }
101               }
102
103               return userData;
104           }
105       }
106
107       //I couldn't figure out how to get the first name, last name, and
            user ID out of my
108       public static class UserSession
109       {
110           public static string CurrentUsername { get; set; } = "";
111           public static bool CheckingAccount { get; set; } //These are
                implemented so that ChooseAccountFrm knows which account to
                get into for withdrawals/deposits
112           public static bool SavingsAccount { get; set; } //These are
                implemented so that ChooseAccountFrm knows which account to
                get into for withdrawals/deposits
113           public static string CurrentFirstName { get; set; } = "";
114           public static string CurrentLastName { get; set; } = "";
115           public static double CurrentCheckingBalance { get; set; }
116           public static double CurrentSavingsBalance { get; set; }
117           public static string CurrentUserID { get; set; } = "";
```

```
118
119         }
120         private bool VerifyLogin(string username, string password)
121         {
122             string allUsersFilePath = Path.Combine
                  (Application.StartupPath, "All_User_Accounts.txt");
123
124             try
125             {
126                 if (File.Exists(allUsersFilePath))
127                 {
128                     // Load data into a 2D array
129                     string[,] userData = UserDataTracking.LoadUserData
                    (allUsersFilePath);
130
131                     // Iterate through the array to find a username and
                    password match
132                     for (int i = 0; i < userData.GetLength(0); i++)
133                     {
134                         string currentUsername = userData[i, 3].Trim();
135                         string currentPassword = userData[i, 4].Trim();
136
137                         if (currentUsername.Equals(username,
                        StringComparison.OrdinalIgnoreCase))
138                         {
139                             if (currentPassword.Equals(password))
140                             {
141
142                                 UserSession.CurrentUserID = userData[i,
                            0].Trim();
143                                 UserSession.CurrentFirstName = userData[i,
                             1].Trim();
144                                 UserSession.CurrentLastName = userData[i,
                            2].Trim();
145                                 UserSession.CurrentCheckingBalance =
                            double.Parse(userData[i, 5].Replace("$", ""));
146                                 UserSession.CurrentSavingsBalance =
                            double.Parse(userData[i, 6].Replace("$", ""));
147
148                                 TransactionLogger.InitializeReceipt();
149
150                                 return true; // Correct username and
                            password match
151                             }
152                             else
153                             {
154                                 MessageBox.Show("Invalid password. Please
                            try again.");
155                                 return false; // Password does not match
```

```
156                                 }
157                             }
158                         }
159                         MessageBox.Show("Username not found. Please try
                        again.");
160                     }
161                 }
162                 catch (Exception ex)
163                 {
164                     MessageBox.Show($"Error checking username and password:
                      {ex.Message}");
165                 }
166                 return false; // Username not found or password does not match
167             }
168 
169 
170             //This is the method I've been using to swap windows everytime a
                  new window is opened.
171             //Without it, the user would have every window open at once.
172             //I implemented this before I knew about public classes so every
                  form has a unique ShowForm.
173             private void ShowForm(Form form)
174             {
175                 this.Hide(); // Hide the main menu
176                 //form.FormClosed += (s, args) => this.Show(); // Show the
                      main menu again when the form is closed
177                 form.Show(); // Show the form
178             }
179 
180             private void FormClosingEvent(object sender, FormClosingEventArgs
                  e)
181             {
182                 // Check if the close reason is user closing the form via the
                      'X' button
183                 if (e.CloseReason == CloseReason.UserClosing)
184                 {
185                     // Ask the user to confirm they want to exit
186                     DialogResult result = MessageBox.Show("Are you sure you
                      want to exit?", "Exit Confirmation",
                      MessageBoxButtons.YesNo, MessageBoxIcon.Question);
187 
188                     // If the user clicks 'No', cancel the closing of the form
189                     if (result == DialogResult.No)
190                     {
191                         e.Cancel = true;
192                     }
193                 }
194                 // If the form is closing for other reasons (like shutdown,
                      system restart, etc.)
```

```
195         }
196
197
198
199         public Form1_LoginFrm()
200         {
201             InitializeComponent();
202
203         }
204
205
206         private void label2_Click(object sender, EventArgs e)
207         {
208
209         }
210
211         private void usernameTxt_TextChanged(object sender, EventArgs e)
212         {
213
214         }
215
216         private void createBtn_Click(object sender, EventArgs e)
217         {
218             ShowForm(new CreateAccountFrm());
219         }
220
221         private void loginBtn_Click(object sender, EventArgs e)
222         {
223             string username = usernameTxt.Text.Trim();
224             string password = passwordTxt.Text.Trim();
225
226             if (VerifyLogin(username, password))
227             {
228                 UserSession.CurrentUsername = username;
229
230                 MessageBox.Show("Login successful!");
231
232                 ShowForm(new mainmenuFrm());
233             }
234             else
235             {
236                 MessageBox.Show("Invalid username/password, please try
                      again.");
237             }
238         }
239
240         private void loginFrm_Load(object sender, EventArgs e)
241         {
242
```

```
243         }
244
245         private void passwordTxt_TextChanged(object sender, EventArgs e)
246         {
247
248         }
249     }
250 }
251
```

ChooseAccountFrm


```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.Data;
 5  using System.Drawing;
 6  using System.Linq;
 7  using System.Text;
 8  using System.Threading.Tasks;
 9  using System.Windows.Forms;
10  using static ISDS309FinalProject.Form1_LoginFrm;
11
12  namespace ISDS309FinalProject
13  {
14      public partial class ChooseAccountFrm : Form
15      {
16          public bool IsWithdrawal { get; set; } //bool that establishes what
               ChooseAccountFrm looks like.
17          public ChooseAccountFrm(bool isWithdrawal)
18          {
19              InitializeComponent();
20              IsWithdrawal = isWithdrawal;
21              this.FormClosing += FormUtilities.CloseFormHandler;
22
23          }
24
25
26
27          private bool checkingAccount = false;
28          private bool savingsAccount = false;
29
30          private void ShowForm(Form form)
31          {
32              this.Hide(); // Hide the main menu
33              form.Show(); // Show the form
34          }
35
36          private void ChooseAccountFrm_FormClosing(object sender,
              FormClosingEventArgs e)
37          {
38              Application.Exit(); // This ensures the entire application
                  exits when the main form attempts to close
39          }
40
41          private void ChooseAccountWithdrawal_Load(object sender, EventArgs
              e)
42          {
43              // Adjusts the label based on the operation
44              this.Text = IsWithdrawal ? "Withdraw" : "Deposit";
45          }
```

```
46
47          private void welcomeLbl_Click(object sender, EventArgs e)
48          {
49
50          }
51
52          //more isWithdrawal logic
53          private void OKBtn_Click(object sender, EventArgs e)
54          {
55
56              if (UserSession.CheckingAccount || UserSession.SavingsAccount)
57              {
58                  Form nextForm = IsWithdrawal ? (Form)new withdrawalFrm() :
                      (Form)new depositFrm();
59                  ShowForm(nextForm);
60              }
61              else
62              {
63                  MessageBox.Show("Please select an account type.");
64              }
65          }
66
67          //establishes the UserSession booleans
68          private void checkingsBtn_CheckedChanged(object sender, EventArgs
              e)
69          {
70              if (checkingsBtn.Checked)
71              {
72                  UserSession.CheckingAccount = true;
73                  UserSession.SavingsAccount = false;
74              }
75          }
76
77          //establishes the UserSession booleans
78          private void savingsBtn_CheckedChanged(object sender, EventArgs e)
79          {
80              if (savingsBtn.Checked)
81              {
82                  UserSession.CheckingAccount = false;
83                  UserSession.SavingsAccount = true;
84              }
85          }
86
87          private void cancelBtn_Click(object sender, EventArgs e)
88          {
89              mainmenuFrm settingsForm = new mainmenuFrm();
90              FormUtilities.ShowForm(this, settingsForm);
91          }
92      }
```

```
93  }
94
```

CreateAccountFrm

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.ComponentModel;
 4 using System.Data;
 5 using System.Drawing;
 6 using System.Linq;
 7 using System.Text;
 8 using System.Threading.Tasks;
 9 using System.Windows.Forms;
10 using static ISDS309FinalProject.Form1_LoginFrm;
11
12 namespace ISDS309FinalProject
13 {
14     public partial class CreateAccountFrm : Form
15     {
16         public CreateAccountFrm()
17         {
18             InitializeComponent();
19         }
20
21         private bool UsernameExists(string username)
22         {
23             string allUsersFilePath = Path.Combine
                 (Application.StartupPath, "All_User_Accounts.txt");
24             try
25             {
26                 if (File.Exists(allUsersFilePath))
27                 {
28                     string[] lines = File.ReadAllLines(allUsersFilePath);
29                     // Skip the header line by starting the loop from
                    index 1
30                     for (int i = 1; i < lines.Length; i++)
31                     {
32                         string[] columns = lines[i].Split(',');
33                         // Checking the Username field (index 3)
34                         if (columns.Length > 3 && columns[4].Trim().Equals
                    (username, StringComparison.OrdinalIgnoreCase))
35                         {
36                             return true; // Username found
37                         }
38                     }
39                 }
40             }
41             catch (Exception ex)
42             {
43                 MessageBox.Show($"Error checking username: {ex.Message}");
44             }
45             return false; // Username not found
46         }
```

```
89              }
90
91              //generates a random user ID
92              string userID = GenerateUserID(20);
93              string filename = $"Account_Information_{firstName}_
                  {lastName}.txt";
94              string filepath = Path.Combine(Application.StartupPath,
                  filename);
95
96              // Add username and password to All_User_Accounts.txt
97              string allUsersFilePath = Path.Combine
                  (Application.StartupPath, "All_User_Accounts.txt");
98              using (StreamWriter writer = new StreamWriter
                  (allUsersFilePath, true))
99              {
100                 // Checks if the file is new or empty and add headers if
                      necessary.
101                 if (new FileInfo(allUsersFilePath).Length == 0)
102                 {
103                     writer.WriteLine("UserID,First Name,Last
                        Name,Username,Password,Checking Account Balance,Savings
                        Account Balance");
104                 }
105
106                 // Write user information in CSV format
107                 writer.WriteLine($"{userID},{firstName},{lastName},
                      {username},{password},${checkingAccountBalance},
                      ${savingsAccountBalance}");
108
109                 MessageBox.Show($"Account created successfully! File
                      saved: {filename}");
110             }
111         }
112
113
114
115         private void label2_Click(object sender, EventArgs e)
116         {
117
118         }
119
120         private void textBox3_TextChanged(object sender, EventArgs e)
121         {
122
123         }
124
125         private void usernameTxt_TextChanged(object sender, EventArgs e)
126         {
127
```

```
128         }
129
130         private void label1_Click(object sender, EventArgs e)
131         {
132
133         }
134
135         private void firstNameInput_TextChanged(object sender, EventArgs
              e)
136         {
137
138         }
139
140         private void button1_Click(object sender, EventArgs e)
141         {
142             Form1_LoginFrm settingsForm = new Form1_LoginFrm();
143             FormUtilities.ShowForm(this, settingsForm);
144         }
145
146         private void CreateAccountFrm_Load(object sender, EventArgs e)
147         {
148
149         }
150     }
151 }
152
```

depositFrm

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.ComponentModel;
 4 using System.Data;
 5 using System.Drawing;
 6 using System.Linq;
 7 using System.Text;
 8 using System.Threading.Tasks;
 9 using System.Windows.Forms;
10 using static ISDS309FinalProject.Form1_LoginFrm;
11
12 namespace ISDS309FinalProject
13 {
14     public partial class depositFrm : Form
15     {
16         public depositFrm()
17         {
18             InitializeComponent();
19             this.FormClosing += FormUtilities.CloseFormHandler;
20         }
21
22         public bool UpdateAccountBalanceDeposit(double amount, out double
            remainingBalance)
23         {
24             string allUsersFilePath = Path.Combine
              (Application.StartupPath, "All_User_Accounts.txt");
25             bool updateSuccessful = false;
26             remainingBalance = 0.0;
27             double currentBalance = 0;
28             string username = UserSession.CurrentUsername;
29             string accountType = UserSession.CheckingAccount ?
              "Checking" : "Savings";
30
31             // Load user data into array
32             string[,] userData = UserDataTracking.LoadUserData
              (allUsersFilePath);
33
34             // Find the user and update the balance
35             for (int i = 1; i < userData.GetLength(0); i++)
36             {
37                 if (userData[i, 3] == username)
38                 {
39                     int balanceColumn = accountType == "Checking" ? 5 : 6;
40                     currentBalance = double.Parse(userData[i,
                    balanceColumn].Trim('$'));
41                     currentBalance += amount;
42                     userData[i, balanceColumn] =
                    $"${currentBalance:0.00}";
43                     updateSuccessful = true;
```

```
44                      remainingBalance = currentBalance;
45                      break;
46                  }
47              }
48
49              // Write updated data back to file
50              if (updateSuccessful)
51              {
52                  using (StreamWriter writer = new StreamWriter
                      (allUsersFilePath, false))  // Overwrite file
53                  {
54                      writer.WriteLine("UserID,First Name,Last
                        Name,Username,Password,Checking Account Balance,Savings
                        Account Balance");
55                      for (int i = 1; i < userData.GetLength(0); i++)
56                      {
57                          writer.WriteLine(string.Join(",", userData[i, 0],
                        userData[i, 1], userData[i, 2], userData[i, 3], userData
                        [i, 4], userData[i, 5], userData[i, 6]));
58                      }
59                  }
60              }
61
62              return updateSuccessful;
63          }
64
65          private void DepositMessage(double amount)
66          {
67              string accountType = UserSession.CheckingAccount ?
                  "Checking" : "Savings";  // Determine the account type
68
                  // Prompt the user to confirm the deposit
69              double currentAccountBalance = UserSession.CheckingAccount ?
                  UserSession.CurrentCheckingBalance :
                  UserSession.CurrentSavingsBalance;
70              DialogResult result = MessageBox.Show($"Are you sure you want
                  to deposit ${amount:0.00} into your {accountType} account?",
                   "Confirm deposit", MessageBoxButtons.YesNo,
                  MessageBoxIcon.Question);
71
72              // Check the user's response
73              if (result == DialogResult.Yes)
74              {
75                  double remainingBalance;
76                  bool wasUpdateSuccessful = UpdateAccountBalanceDeposit
                      (amount, out remainingBalance);
77                  if (UserSession.CheckingAccount)
78                  {
79                      UserSession.CurrentCheckingBalance = remainingBalance;
```

```
80                      TransactionLogger.LogTransaction("Deposit", amount,
                      "Checking", remainingBalance);
81
82                  }
83                  else
84                  {
85                      UserSession.CurrentSavingsBalance = remainingBalance;
86                      TransactionLogger.LogTransaction("Deposit", amount,
                      "Savings", remainingBalance);
87                  }
88                  MessageBox.Show($"Successfully deposited ${amount} into
                    your account!");
89              }
90              else
91              {
92                  MessageBox.Show("deposit cancelled.", "Transaction
                    Cancelled", MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
93              }
94          }
95
96          private void cancelBtn_Click(object sender, EventArgs e)
97          {
98              mainmenuFrm settingsForm = new mainmenuFrm();
99              FormUtilities.ShowForm(this, settingsForm);
100
101         }
102
103         private void depositFrm_Load(object sender, EventArgs e)
104         {
105
106         }
107
108         private void fortyBtn_Click_1(object sender, EventArgs e)
109         {
110             DepositMessage(40.00);
111         }
112
113         private void eightyBtn_Click_1(object sender, EventArgs e)
114         {
115             DepositMessage(80.00);
116         }
117
118         private void hundredBtn_Click_1(object sender, EventArgs e)
119         {
120             DepositMessage(100.00);
121         }
122
123         private void twohundredBtn_Click_1(object sender, EventArgs e)
```

withdrawalFrm

...ct-master\ISDS309FinalProject-master\withdrawalFrm.cs 1

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.Data;
 5  using System.Drawing;
 6  using System.Linq;
 7  using System.Text;
 8  using System.Threading.Tasks;
 9  using System.Windows.Forms;
10  using static ISDS309FinalProject.Form1_LoginFrm;
11  using static
      System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;
12
13  namespace ISDS309FinalProject
14  {
15      public partial class withdrawalFrm : Form
16      {
17          public withdrawalFrm()
18          {
19              InitializeComponent();
20              this.FormClosing += FormUtilities.CloseFormHandler;
21          }
22
23          public bool UpdateAccountBalanceWithdraw(double amount, out double
               remainingBalance)
24          {
25              string allUsersFilePath = Path.Combine
                  (Application.StartupPath, "All_User_Accounts.txt");
26              bool updateSuccessful = false;
27              remainingBalance = 0.0;
28              double currentBalance = 0;
29              string username = UserSession.CurrentUsername;
30              string accountType = UserSession.CheckingAccount ?
                  "Checking" : "Savings";
31
32              // Load user data into array
33              string[,] userData = UserDataTracking.LoadUserData
                  (allUsersFilePath);
34
35              // Find the user and update the balance
36              for (int i = 1; i < userData.GetLength(0); i++)
37              {
38                  if (userData[i, 3].Trim().Equals(username,
                      StringComparison.OrdinalIgnoreCase))  // Check the
                      username match
39                  {
40                      int balanceColumn = accountType == "Checking" ? 5 :
                      6;  // Adjust indices based on actual column positions
41                      currentBalance = double.Parse(userData[i,
```

```
                    balanceColumn].Trim('$'));
42                   if (currentBalance >= amount)
43                   {
44                       currentBalance -= amount;
45                       userData[i, balanceColumn] =
                    $"${currentBalance:0.00}";
46                       updateSuccessful = true;
47                       remainingBalance = currentBalance;
48                       break;  // Exit the loop once update is done
49                   }
50                   else
51                   {
52                       MessageBox.Show($"Insufficient funds for this
                    transaction. Available funds: ${currentBalance:0.00}");
53                       return false;  // Exit method early if
                    insufficient funds
54                   }
55               }
56           }
57
58           // Write updated data back to file if update was successful
59           if (updateSuccessful)
60           {
61               using (StreamWriter writer = new StreamWriter
                   (allUsersFilePath, false))  // Overwrite the entire file
62               {
63                   writer.WriteLine("UserID,First Name,Last
                    Name,Username,Password,Checking Account Balance,Savings
                    Account Balance");
64                   for (int j = 0; j < userData.GetLength(0); j++)  //
                    Start from 0 to include headers
65                   {
66                       writer.WriteLine(string.Join(",", userData[j, 0],
                    userData[j, 1], userData[j, 2], userData[j, 3], userData
                    [j, 4], userData[j, 5], userData[j, 6]));
67                   }
68               }
69           }
70
71           return updateSuccessful;
72       }
73
74
75       private void WithdrawalMessage(double amount)
76       {
77           string accountType = UserSession.CheckingAccount ?
                "Checking" : "Savings";  // Determine the account type
78
                // Prompt the user to confirm the withdrawal
```

```
79              double currentAccountBalance = UserSession.CheckingAccount ?
                  UserSession.CurrentCheckingBalance :
                  UserSession.CurrentSavingsBalance;
80              DialogResult result = MessageBox.Show($"Are you sure you want
                  to withdraw ${amount:0.00} from your {accountType}
                  account?", "Confirm Withdrawal", MessageBoxButtons.YesNo,
                  MessageBoxIcon.Question);
81
82              // Check the user's response
83              if (result == DialogResult.Yes)
84              {
85                  double remainingBalance;
86                  bool wasUpdateSuccessful = UpdateAccountBalanceWithdraw
                      (amount, out remainingBalance);
87                  if (wasUpdateSuccessful)
88                  {
89                      if (UserSession.CheckingAccount)
90                      {
91                          UserSession.CurrentCheckingBalance =
                        remainingBalance;
92                          TransactionLogger.LogTransaction("Withdrawal",
                        amount, "Checking", remainingBalance);
93                      }
94                      else
95                      {
96                          UserSession.CurrentSavingsBalance =
                        remainingBalance;
97                          TransactionLogger.LogTransaction("Withdrawal",
                        amount, "Savings", remainingBalance);
98                      }
99                      // Show success message only if the update was
                        successful
100                     MessageBox.Show($"Successfully withdrew ${amount} from
                         your account! Remaining balance:
                        ${remainingBalance:0.00}", "Withdrawal Successful",
                        MessageBoxButtons.OK, MessageBoxIcon.Information);
101                 }
102                 else
103                 {
104                     // Show error message if the withdrawal was not
                        successful
105                     MessageBox.Show($"Withdrawal failed. Please check the
                        amount and try again.", "Withdrawal Failed",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
106                 }
107             }
108             else
109             {
110                 MessageBox.Show("Withdrawal cancelled.", "Transaction
```

```
                    Cancelled", MessageBoxButtons.OK,
                    MessageBoxIcon.Information);
            }
        }




        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void fortyBtn_Click(object sender, EventArgs e)
        {
            WithdrawalMessage(40.00);
        }

        private void eightyBtn_Click(object sender, EventArgs e)
        {
            WithdrawalMessage(80.00);
        }

        private void hundredBtn_Click(object sender, EventArgs e)
        {
            WithdrawalMessage(100.00);
        }

        private void twohundredBtn_Click(object sender, EventArgs e)
        {
            WithdrawalMessage(200.00);
        }

        private void otherBtn_Click(object sender, EventArgs e)
        {
            string input = Microsoft.VisualBasic.Interaction.InputBox
              ("Enter the amount to withdraw:", "Withdraw", "0", -1, -1);
            if (double.TryParse(input, out double customAmount) &&
              customAmount > 0)
            {
                WithdrawalMessage(customAmount);
            }
            else
            {
                MessageBox.Show("Invalid amount entered.");
            }
        }

        private void OKBtn_Click(object sender, EventArgs e)
```

```
156         {
157
158         }
159
160         private void cancelBtn_Click(object sender, EventArgs e)
161         {
162             mainmenuFrm settingsForm = new mainmenuFrm();
163             FormUtilities.ShowForm(this, settingsForm);
164         }
165     }
166 }
167
```

balanceFrm

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.ComponentModel;
 4  using System.Data;
 5  using System.Drawing;
 6  using System.Linq;
 7  using System.Text;
 8  using System.Threading.Tasks;
 9  using System.Windows.Forms;
10  using static ISDS309FinalProject.Form1_LoginFrm;
11
12  namespace ISDS309FinalProject
13  {
14      public partial class balanceFrm : Form
15      {
16          public balanceFrm()
17          {
18              InitializeComponent();
19              this.FormClosing += FormUtilities.CloseFormHandler;
20              UpdateBalanceLabels();
21          }
22
23          private void ShowForm(Form form)
24          {
25              this.Hide(); // Hide the main menu
26              form.FormClosed += (s, args) => this.Show(); // Show the main
                  menu again when the form is closed
27              form.Show(); // Show the form
28          }
29
30          private void checkingsBtn_CheckedChanged(object sender, EventArgs
                e)
31          {
32
33          }
34
35          private void label2_Click(object sender, EventArgs e)
36          {
37
38          }
39
40          private void balanceFrm_Load(object sender, EventArgs e)
41          {
42
43          }
44
45          private void cancelBtn_Click(object sender, EventArgs e)
46          {
47              ShowForm(new mainmenuFrm());
```

```
48          }
49
50          private void label3_Click(object sender, EventArgs e)
51          {
52
53          }
54
55          private void label4_Click(object sender, EventArgs e)
56          {
57
58          }
59
60          private void UpdateBalanceLabels()
61          {
62              // Assuming UserSession holds the current balances or has
                  methods to fetch them
63              label3.Text =
                  $"${UserSession.CurrentCheckingBalance:0.00}";  // Format as
                  currency
64              label4.Text =
                  $"${UserSession.CurrentSavingsBalance:0.00}";   // Format as
                  currency
65          }
66      }
67  }
68
```

transferFrm

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.ComponentModel;
 4 using System.Data;
 5 using System.Drawing;
 6 using System.Linq;
 7 using System.Text;
 8 using System.Threading.Tasks;
 9 using System.Windows.Forms;
10 using static ISDS309FinalProject.Form1_LoginFrm;
11
12 namespace ISDS309FinalProject
13 {
14     public partial class transferFrm : Form
15     {
16         private withdrawalFrm withdrawalForm = new withdrawalFrm();
17         private depositFrm depositForm = new depositFrm();
18         public transferFrm()
19         {
20             InitializeComponent();
21             InitializeAccountBoxes();
22             this.FormClosing += FormUtilities.CloseFormHandler;
23         }
24
25         private void InitializeAccountBoxes()
26         {
27
28             accountBox.Items.Add("Checkings");
29             accountBox.Items.Add("Savings");
30
31             accountBox2.Items.Add("Checkings");
32             accountBox2.Items.Add("Savings");
33
34             accountBox.SelectedIndex = 0;  // Default to Checking
35             accountBox2.SelectedIndex = 1;  // Default to Savings
36         }
37
38         private void accountBox_SelectedIndexChanged(object sender,
              EventArgs e)
39         {
40
41         }
42
43         private void pictureBox1_Click(object sender, EventArgs e)
44         {
45
46         }
47
48         private void transferFrm_Load(object sender, EventArgs e)
```

```
49          {
50
51          }
52
53          private void toLbl_Click(object sender, EventArgs e)
54          {
55
56          }
57
58          private void accountBox2_SelectedIndexChanged(object sender,
              EventArgs e)
59          {
60
61          }
62
63          private void OKBtn_Click(object sender, EventArgs e)
64          {
65              double amount;
66              double remainingBalanceWithdraw;
67              double remainingBalanceDeposit;
68
69              if (!double.TryParse(transferAmt.Text, out amount))
70              {
71                  MessageBox.Show("Please enter a valid amount.", "Invalid
                      Input", MessageBoxButtons.OK, MessageBoxIcon.Warning);
72                  return;
73              }
74
75              if (accountBox.Text == accountBox2.Text)
76              {
77                  MessageBox.Show("You cannot transfer between the same
                      account.", "Error", MessageBoxButtons.OK,
                      MessageBoxIcon.Error);
78                  return;
79              }
80
81              bool withdrawFromChecking = accountBox.Text == "Checkings";
82              bool depositToChecking = accountBox2.Text == "Checkings";
83
84              // Adjust UserSession flags for withdrawal
85              UserSession.CheckingAccount = withdrawFromChecking;
86              UserSession.SavingsAccount = !withdrawFromChecking;
87
88              // Perform withdrawal from the first account
89              if (withdrawalForm.UpdateAccountBalanceWithdraw(amount, out
                  remainingBalanceWithdraw))
90              {
91                  // Adjust UserSession flags for deposit
92                  UserSession.CheckingAccount = depositToChecking;
```

```
 93                  UserSession.SavingsAccount = !depositToChecking;
 94
 95                  // Perform deposit to the second account
 96                  if (depositForm.UpdateAccountBalanceDeposit(amount, out
                       remainingBalanceDeposit))
 97                  {
 98                      // Update session balances
 99                      if (withdrawFromChecking)
100                      {
101                          UserSession.CurrentCheckingBalance =
                        remainingBalanceWithdraw;
102                          TransactionLogger.LogTransaction("Transfer",
                        amount, "Checking => Savings",
                        remainingBalanceWithdraw);
103                      }
104                      else
105                      {
106                          UserSession.CurrentSavingsBalance =
                        remainingBalanceWithdraw;
107                          TransactionLogger.LogTransaction("Transfer",
                        amount, "Savings => Checking",
                        remainingBalanceWithdraw);
108                      }
109
110                      if (depositToChecking)
111                      {
112                          UserSession.CurrentCheckingBalance =
                        remainingBalanceDeposit;
113
114                      }
115                      else
116                      {
117                          UserSession.CurrentSavingsBalance =
                        remainingBalanceDeposit;
118
119                      }
120
121                      MessageBox.Show($"Transfer successful. New balance in
                        the target account: ${remainingBalanceDeposit:0.00}",
                        "Success", MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
122                  }
123                  else
124                  {
125                      MessageBox.Show("Transfer failed during deposit.",
                        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
126                  }
127              }
128              else
```

```
129             {
130                 MessageBox.Show("Transfer failed during withdrawal.",
                      "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
131             }
132         }
133
134         private void label1_Click(object sender, EventArgs e)
135         {
136
137         }
138
139         private void cancelBtn_Click(object sender, EventArgs e)
140         {
141             mainmenuFrm settingsForm = new mainmenuFrm();
142             FormUtilities.ShowForm(this, settingsForm);
143         }
144
145         private void transferAmt_TextChanged(object sender, EventArgs e)
146         {
147
148         }
149     }
150 }
151
```

Program.cs File

...lProject-master\ISDS309FinalProject-master\Program.cs 1

```
1  using static ISDS309FinalProject.Form1_LoginFrm;
2
3  namespace ISDS309FinalProject
4  {
5      internal static class Program
6      {
7          /// <summary>
8          ///  The main entry point for the application.
9          /// </summary>
10         [STAThread]
11         static void Main()
12         {
13             // To customize application configuration such as set high DPI
                settings or default font,
14             // see https://aka.ms/applicationconfiguration.
15
16             //I needed to adjust this so that FinalizeReceipt doesn't
                activate as many times as there are windows open, I googled
                how to do this.
17             ApplicationConfiguration.Initialize();
18             Application.EnableVisualStyles();
19             Application.SetCompatibleTextRenderingDefault(true);
20             Application.ApplicationExit += new EventHandler
                (Application_ApplicationExit);
21             Application.Run(new Form1_LoginFrm());
22         }
23
24         private static void Application_ApplicationExit(object sender,
            EventArgs e)
25         {
26             TransactionLogger.FinalizeReceipt(); // Handles final receipt
                writing here
27         }
28     }
29 }
```