***Visual Programming Lab***

***CSL-313***

***Lab Journal   
( 2+3 )  
Complete Package***

****

**Student Name: M. Anas Baig**

**Enrolment No.: 01-134152-037**

**Class and Section: BS(CS)-5A**

**Department of Computer Science**

**BAHRIA UNIVERSITY, ISLAMABAD**

My Virtual Bank

**In the lab task you will simulate a virtual Bank just like traditional Bank using OOP concepts and C# Console Application. Implementation details are as follows;**

1. Account class with attributes {AccountNO, AccountTitle, CNIC, ContactNumber, Balance}
2. Two types of account 1. Current Account with attribute {WithdrawalLimit}, 2. Saving Account with attribute {ProfitPercentage}; both types should be inherited from the Account Class.
3. Program should allow user to open a new account with all information of account holder and type of account he wants to open. (CRUD Operations)
4. You are required to enable an account holder to perform transactions withdrawal and deposit.
5. Validate your inputs; especially for withdrawal and deposit.

**Procedure/Program:**

**program.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.IO;
6. **using** System.Collections;
8. **namespace** ConsoleApplication1
9. {
10. **class** program
11. {
12. **static** **void** Main(**string**[] args)
13. {
14. Console.BackgroundColor = ConsoleColor.Gray;
15. Console.Clear();
16. Console.ForegroundColor = ConsoleColor.Black;
17. Console.WriteLine("===============================================================================");
18. Console.WriteLine("   B A H R I A - U N I V E R S I T Y - V I R T U A L - B A N K - S Y S T E M");
19. Console.WriteLine("===============================================================================");
20. consoleMenu c = **new** consoleMenu();
21. c.startMenu();
22. Console.ReadLine();
23. }
24. }
25. }

**account.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** ConsoleApplication1
7. {
8. **class** account
9. {
10. **protected** **string** accountNo;
11. **protected** **string** accountTitle;
12. **protected** **string** cnic;
13. **protected** **string** contactNo;
14. **protected** **double** balance;
16. **public** account()
17. {
18. }
20. **public** account(**string** accountNo, **string** accountTitle, **string** cnic, **string** contactNo, **double** balance)
21. {
22. **this**.accountNo = accountNo;
23. **this**.accountTitle = accountTitle;
24. **this**.cnic = cnic;
25. **this**.contactNo = contactNo;
26. **this**.balance = balance;
27. }
29. **public** **virtual** **double** withdraw(**double** amount) //virtual function
30. {
31. **return** (**this**.balance -= amount);
32. }
34. **public** **virtual** **double** deposit(**double** amount) //virtual function
35. {
36. **return** (**this**.balance += amount);
37. }
38. }
39. }

**currentAccount.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** ConsoleApplication1
7. {
8. **class** currentAccount : account //inheritence
9. {
10. **private** **double** withdrawalLimit; //extra variable to store withdrawal limit
12. **public** currentAccount()
13. {
14. }
16. **public** currentAccount( **string** accountNo, **string** accountTitle, **string** cnic, **string** contactNo, **double** balance, **double** withdrawalLimit )
17. : **base**(accountNo, accountTitle, cnic, contactNo, balance) //base class parametrized constructor call combine with child class
18. {
19. **this**.withdrawalLimit = withdrawalLimit;
20. }
22. **public** **string** accountNoProperty //C# property: just like get set method
23. {
24. **get** { **return** accountNo; }
25. **set** { accountNo = value; }
26. }
28. **public** **string** accountTitleProperty //C# property: just like get set method
29. {
30. **get** { **return** accountTitle; }
31. **set** { accountTitle = value; }
32. }
34. **public** **string** cnicProperty //C# property: just like get set method
35. {
36. **get** { **return** cnic; }
37. **set** { cnic = value; }
38. }
40. **public** **string** contactNoProperty //C# property: just like get set method
41. {
42. **get** { **return** contactNo; }
43. **set** { contactNo = value; }
44. }
46. **public** **double** balanceProperty //C# property: just like get set method
47. {
48. **get** { **return** balance; }
49. **set** { balance = value; }
50. }
52. **public** **double** withdrawalLimitProperty //C# property: just like get set method
53. {
54. **get** { **return** withdrawalLimit; }
55. **set** { withdrawalLimit = value; }
56. }
58. **public** **override** **double** deposit(**double** amount) //virtual funtion overridden in child class
59. {
60. **return** **base**.deposit(amount);
61. }
63. **public** **override** **double** withdraw(**double** amount) //virtual funtion overridden in child class
64. {
65. **if** (amount <= **this**.withdrawalLimit)
66. {
67. **return** **base**.withdraw(amount);
68. }
69. **return** -1;
70. }
71. }
72. }

**savingAccount.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** ConsoleApplication1
7. {
8. **class** savingAccount : account //inheritence
9. {
10. **private** **float** profitPercentage; //extra variable to store profit percentage
12. **public** savingAccount()
13. {
14. }
16. **public** savingAccount( **string** accountNo, **string** accountTitle, **string** cnic, **string** contactNo, **double** balance, **float** profitPercentage )
17. : **base**(accountNo, accountTitle, cnic, contactNo, balance) //base class parametrized constructor call combine with child class
18. {
19. **this**.profitPercentage = profitPercentage;
20. }
22. **public** **string** accountNoProperty //C# property: just like get set method
23. {
24. **get** { **return** accountNo; }
25. **set** { accountNo = value; }
26. }
28. **public** **string** accountTitleProperty //C# property: just like get set method
29. {
30. **get** { **return** accountTitle; }
31. **set** { accountTitle = value; }
32. }
34. **public** **string** cnicProperty //C# property: just like get set method
35. {
36. **get** { **return** cnic; }
37. **set** { cnic = value; }
38. }
40. **public** **string** contactNoProperty //C# property: just like get set method
41. {
42. **get** { **return** contactNo; }
43. **set** { contactNo = value; }
44. }
46. **public** **double** balanceProperty //C# property: just like get set method
47. {
48. **get** { **return** balance; }
49. **set** { balance = value; }
50. }
52. **public** **float** profitPercentageProperty //C# property: just like get set method
53. {
54. **get** { **return** profitPercentage; }
55. **set** { profitPercentage = value; }
56. }
58. **public** **override** **double** deposit(**double** amount) //virtual funtion overridden in child class
59. {
60. **if** (amount >= 10000)
61. {
62. profitPercentage++;
63. }
64. **return** **base**.deposit(amount);
65. }
67. **public** **override** **double** withdraw(**double** amount) //virtual funtion overridden in child class
68. {
69. **if** (amount >= 10000)
70. {
71. profitPercentage--;
72. }
73. **return** **base**.withdraw(amount);
74. }
75. }
76. }

**consoleMenu.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.IO;
6. **using** System.Collections;
8. **namespace** ConsoleApplication1
9. {
10. **class** consoleMenu
11. {
12. accountManager manager = **new** accountManager();
14. **public** **void** startMenu()
15. {
16. printOptions();
17. **int** num = **int**.Parse(Console.ReadLine());
18. **if**( num == 1 )
19. {
20. accountCreation();
21. }
22. **if**( num == 2 )
23. {
24. searchAccount();
25. }
26. **if** (num == 3)
27. {
28. updateAccount();
29. }
30. **if** (num == 4)
31. {
32. deleteAccount();
33. }
34. **if** (num == 5)
35. {
36. displayAccount();
37. }
38. **if** (num == 6)
39. {
40. transaction();
41. }
42. }
44. **public** **void** printOptions()
45. {
46. Console.WriteLine("Enter your desired operation:");
47. Console.WriteLine("1. Create New Account.");
48. Console.WriteLine("2. Search Account by Account Number.");
49. Console.WriteLine("3. Update Account by Account Number.");
50. Console.WriteLine("4. Delete Account by Account Number.");
51. Console.WriteLine("5. Display Account by Account Number.");
52. Console.WriteLine("6. Deposit / Withdraw by Account Number.");
53. }
55. **public** **void** accountCreation()
56. {
57. **int** num;
58. **do**
59. {
60. Console.Write("\nEnter Account Number:\n");
61. **string** An = Console.ReadLine();
63. Console.Write("\nEnter Account Title:\n");
64. **string** At = Console.ReadLine();
66. Console.Write("\nEnter CNIC:\n");
67. **string** cnic = Console.ReadLine();
69. Console.Write("\nEnter Contact Number:\n");
70. **string** cn = Console.ReadLine();
72. Console.Write("\nEnter Account Balance:\n");
73. **double** bal = **double**.Parse(Console.ReadLine());
75. Console.Write("\nChoose Account Type:\n");
76. Console.Write("1.Saving Account.    2.Current Account.\n");
77. **int** option = **int**.Parse(Console.ReadLine());
79. **if** (option == 1)
80. {
81. Console.WriteLine("\nEnter Profit Percentage:");
82. **float** pp = **float**.Parse(Console.ReadLine());
84. savingAccount savingAcc = **new** savingAccount(An, At, cnic, cn, bal, pp);
85. manager.createNewSavingAccount(savingAcc);
86. }
87. **if** (option == 2)
88. {
89. Console.WriteLine("\nEnter Withdrawal Limit:");
90. **double** wl = **double**.Parse(Console.ReadLine());
92. currentAccount currentAcc = **new** currentAccount(An, At, cnic, cn, bal, wl);
93. manager.createNewCurrentAccount(currentAcc);
94. }
95. Console.WriteLine("\nPress 1 to continue creation of Accounts.");
96. num = **int**.Parse(Console.ReadLine());
97. }
98. **while** (num == 1);
99. manager.writeAllinFile();
100. }
102. **public** **void** searchAccount()
103. {
104. Console.Write("\nChoose Account Type:\n");
105. Console.Write("1.Current Account.    2.Saving Account.\n");
106. **int** num = **int**.Parse(Console.ReadLine());
108. **if** ( num == 1)
109. {
110. Console.WriteLine("\nEnter Account Number:");
111. **string** accnum = Console.ReadLine();
112. manager.searchCurrentAccount(accnum);
113. }
114. **else** **if**( num  == 2)
115. {
116. Console.WriteLine("\nEnter Account Number:");
117. **string** accnum = Console.ReadLine();
118. manager.searchSavingAccount(accnum);
119. }
120. **else**
121. {
122. Console.WriteLine("Invalid Input\n");
123. }
124. }
126. **public** **void** updateAccount()
127. {
128. Console.Write("\nChoose Account Type:\n");
129. Console.Write("1.Current Account.   2.Saving Account.\n");
130. **int** num = **int**.Parse(Console.ReadLine());
132. **if** (num == 1)
133. {
134. Console.WriteLine("\nEnter Account Number:");
135. **string** accnum = Console.ReadLine();
137. accountFile obj = **new** accountFile();
138. List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
139. **int** totalAccounts = accounts.Count; //built-in count list function
140. **bool** found = **false**;
142. **for** (**int** i = 0; i < totalAccounts; i++)
143. {
144. **if** (accounts[i].accountNoProperty == accnum)
145. {
146. found = **true**;
147. Console.WriteLine("\nAccount Found.\n");
149. Console.Write("\nEnter New Account Number:\n");
150. accounts[i].accountNoProperty = Console.ReadLine();
152. Console.Write("\nEnter New Account Title:\n");
153. accounts[i].accountTitleProperty = Console.ReadLine();
155. Console.Write("\nEnter New CNIC:\n");
156. accounts[i].cnicProperty = Console.ReadLine();
158. Console.Write("\nEnter New Contact Number:\n");
159. accounts[i].contactNoProperty = Console.ReadLine();
161. Console.Write("\nEnter New Account Balance:\n");
162. accounts[i].balanceProperty = **double**.Parse(Console.ReadLine());
164. Console.WriteLine("\nEnter New Withdrawal Limit:");
165. accounts[i].withdrawalLimitProperty = **double**.Parse(Console.ReadLine());
166. }
167. }
169. StreamWriter writeCurrent = **new** StreamWriter("CurrentAccount.txt"); //write to file object
170. //int totalAccount = accounts.Count;
171. **for** (**int** j = 0; j < totalAccounts; j++)
172. {
173. currentAccount account = accounts[j];
174. writeCurrent.WriteLine(account.accountNoProperty);
175. writeCurrent.WriteLine(account.accountTitleProperty);
176. writeCurrent.WriteLine(account.balanceProperty);
177. writeCurrent.WriteLine(account.cnicProperty);
178. writeCurrent.WriteLine(account.contactNoProperty);
179. writeCurrent.WriteLine(account.withdrawalLimitProperty);
180. }
181. writeCurrent.Close();
182. **return**;
183. }
185. **else** **if** (num == 2)
186. {
187. Console.WriteLine("\nEnter Account Number:");
188. **string** accnum = Console.ReadLine();
190. accountFile obj = **new** accountFile();
191. ArrayList accounts = obj.readAllSavingAccount(); //array list: saves diffeent types of objects
192. //List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
193. **int** totalAccounts = accounts.Count; //built-in count list function
194. **bool** found = **false**;
196. **for** (**int** i = 0; i < totalAccounts; i++)
197. {
198. **if** ((accounts[i] **as** savingAccount).accountNoProperty == accnum)
199. {
200. found = **true**;
201. Console.WriteLine("\nAccount Found.\n");
203. Console.Write("\nEnter New Account Number:\n");
204. (accounts[i] **as** savingAccount).accountNoProperty = Console.ReadLine();
206. Console.Write("\nEnter New Account Title:\n");
207. (accounts[i] **as** savingAccount).accountTitleProperty = Console.ReadLine();
209. Console.Write("\nEnter New CNIC:\n");
210. (accounts[i] **as** savingAccount).cnicProperty = Console.ReadLine();
212. Console.Write("\nEnter New Contact Number:\n");
213. (accounts[i] **as** savingAccount).contactNoProperty = Console.ReadLine();
215. Console.Write("\nEnter New Account Balance:\n");
216. (accounts[i] **as** savingAccount).balanceProperty = **double**.Parse(Console.ReadLine());
218. Console.WriteLine("\nEnter New Profit Percentage:");
219. (accounts[i] **as** savingAccount).profitPercentageProperty = **float**.Parse(Console.ReadLine());
220. }
221. }
223. StreamWriter writeSaving = **new** StreamWriter("SavingAccount.txt"); //write to file object
224. //int totalAccount = accounts.Count;
225. **for** (**int** j = 0; j < totalAccounts; j++)
226. {
227. savingAccount account = accounts[j] **as** savingAccount;
228. writeSaving.WriteLine(account.accountNoProperty);
229. writeSaving.WriteLine(account.accountTitleProperty);
230. writeSaving.WriteLine(account.balanceProperty);
231. writeSaving.WriteLine(account.cnicProperty);
232. writeSaving.WriteLine(account.contactNoProperty);
233. writeSaving.WriteLine(account.profitPercentageProperty);
234. }
235. writeSaving.Close();
236. **return**;
237. }
238. **else**
239. {
240. Console.WriteLine("Invalid Input\n");
241. }
242. }
244. **public** **void** deleteAccount()
245. {
246. Console.Write("\nChoose Account Type:\n");
247. Console.Write("1.Current Account.   2.Saving Account.\n");
248. **int** num = **int**.Parse(Console.ReadLine());
250. **if** (num == 1)
251. {
252. Console.WriteLine("\nEnter Account Number:");
253. **string** accnum = Console.ReadLine();
254. accountFile obj = **new** accountFile();
255. List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
256. **int** totalAccounts = accounts.Count; //built-in count list function
257. **bool** found = **false**;
259. **for** (**int** i = 0; i < totalAccounts; i++)
260. {
261. **if** (accounts[i].accountNoProperty == accnum)
262. {
263. found = **true**;
264. Console.WriteLine("\nAccount Found.\n");
266. accounts.Remove(accounts[i]); //deletes that object with i index
267. totalAccounts--;
268. }
269. }
271. StreamWriter writeCurrent = **new** StreamWriter("CurrentAccount.txt"); //write to file object
273. **for** (**int** j = 0; j < totalAccounts; j++)
274. {
275. currentAccount account = accounts[j];
276. writeCurrent.WriteLine(account.accountNoProperty);
277. writeCurrent.WriteLine(account.accountTitleProperty);
278. writeCurrent.WriteLine(account.balanceProperty);
279. writeCurrent.WriteLine(account.cnicProperty);
280. writeCurrent.WriteLine(account.contactNoProperty);
281. writeCurrent.WriteLine(account.withdrawalLimitProperty);
282. }
283. writeCurrent.Close();
284. **return**;
285. }
287. **else** **if** (num == 2)
288. {
289. Console.WriteLine("\nEnter Account Number:");
290. **string** accnum = Console.ReadLine();
291. accountFile obj = **new** accountFile();
292. ArrayList accounts = obj.readAllSavingAccount(); //array list: saves diffeent types of objects
293. **int** totalAccounts = accounts.Count; //built-in count list function
294. **bool** found = **false**;
296. **for** (**int** i = 0; i < totalAccounts; i++)
297. {
298. **if** ((accounts[i] **as** savingAccount).accountNoProperty == accnum)
299. {
300. found = **true**;
301. Console.WriteLine("\nAccount Found.\n");
303. accounts.Remove(accounts[i]); //deletes that object with i index
304. totalAccounts--;
305. }
306. }
308. StreamWriter writeSaving = **new** StreamWriter("SavingAccount.txt"); //write to file object
310. **for** (**int** j = 0; j < totalAccounts; j++)
311. {
312. savingAccount account = accounts[j] **as** savingAccount;
313. writeSaving.WriteLine(account.accountNoProperty);
314. writeSaving.WriteLine(account.accountTitleProperty);
315. writeSaving.WriteLine(account.balanceProperty);
316. writeSaving.WriteLine(account.cnicProperty);
317. writeSaving.WriteLine(account.contactNoProperty);
318. writeSaving.WriteLine(account.profitPercentageProperty);
319. }
320. writeSaving.Close();
321. **return**;
322. }
323. **else**
324. {
325. Console.WriteLine("Invalid Input\n");
326. }
327. }

330. **public** **void** displayAccount()
331. {
332. Console.Write("\nChoose Account Type:\n");
333. Console.Write("1.Current Account.   2.Saving Account.\n");
334. **int** num = **int**.Parse(Console.ReadLine());
336. **if** (num == 1)
337. {
338. Console.WriteLine("\nEnter Account Number:");
339. **string** accnum = Console.ReadLine();
341. accountFile obj = **new** accountFile();
342. List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
343. **int** totalAccounts = accounts.Count; //built-in count list function
344. **bool** found = **false**;
346. **for** (**int** i = 0; i < totalAccounts; i++)
347. {
348. **if** (accounts[i].accountNoProperty == accnum)
349. {
350. found = **true**;
351. Console.WriteLine("\nAccount Found.\n");
353. Console.Write("\nAccount Number:    ");
354. Console.Write(accounts[i].accountNoProperty);
356. Console.Write("\nAccount Title: ");
357. Console.Write(accounts[i].accountTitleProperty);
359. Console.Write("\nCNIC:  ");
360. Console.Write(accounts[i].cnicProperty);
362. Console.Write("\nContact Number:    ");
363. Console.Write(accounts[i].contactNoProperty);
365. Console.Write("\nAccount Balance:   ");
366. Console.Write(accounts[i].balanceProperty);
368. Console.WriteLine("\nWithdrawal Limit:  ");
369. Console.Write(accounts[i].withdrawalLimitProperty);
370. }
371. }
372. }
374. **else** **if** (num == 2)
375. {
376. Console.WriteLine("\nEnter Account Number:");
377. **string** accnum = Console.ReadLine();
379. accountFile obj = **new** accountFile();
380. ArrayList accounts = obj.readAllSavingAccount(); //array list: saves diffeent types of objects
382. **int** totalAccounts = accounts.Count; //built-in count list function
383. **bool** found = **false**;
385. **for** (**int** i = 0; i < totalAccounts; i++)
386. {
387. **if** ((accounts[i] **as** savingAccount).accountNoProperty == accnum)
388. {
389. found = **true**;
390. Console.WriteLine("\nAccount Found.\n");
392. Console.Write("\nAccount Number:");
393. Console.Write((accounts[i] **as** savingAccount).accountNoProperty);
395. Console.Write("\nAccount Title:");
396. Console.Write((accounts[i] **as** savingAccount).accountTitleProperty);
398. Console.Write("\nCNIC:");
399. Console.Write((accounts[i] **as** savingAccount).cnicProperty);
401. Console.Write("\nContact Number:");
402. Console.Write((accounts[i] **as** savingAccount).contactNoProperty);
404. Console.Write("\nAccount Balance:");
405. Console.Write((accounts[i] **as** savingAccount).balanceProperty);
407. Console.WriteLine("\nProfit Percentage:");
408. Console.Write((accounts[i] **as** savingAccount).profitPercentageProperty);
409. }
410. }
411. }
412. **else**
413. {
414. Console.WriteLine("Invalid Input\n");
415. }
416. }
418. **public** **void** transaction()
419. {
420. Console.Write("\nChoose Account Type:\n");
421. Console.Write("1.Current Account.   2.Saving Account.\n");
422. **int** num = **int**.Parse(Console.ReadLine());
424. **if** (num == 1)
425. {
426. Console.WriteLine("\nEnter Account Number:");
427. **string** accnum = Console.ReadLine();
429. accountFile obj = **new** accountFile();
430. List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
431. **int** totalAccounts = accounts.Count; //built-in count list function
432. **bool** found = **false**;
434. **for** (**int** i = 0; i < totalAccounts; i++)
435. {
436. **if** (accounts[i].accountNoProperty == accnum)
437. {
438. found = **true**;
439. Console.WriteLine("\nAccount Found.\n");
441. Console.Write("\nChoose Transaction Type:\n");
442. Console.Write("1.Deposit Amount.   2.Withdraw Amount.\n");
443. **int** option = **int**.Parse(Console.ReadLine());
445. **if** (option == 1)
446. {
447. Console.Write("Enter Deposit Amount:\n");
448. **int** depositAmount = **int**.Parse(Console.ReadLine());
449. accounts[i].balanceProperty = (accounts[i].balanceProperty + depositAmount);
450. accounts[i].deposit(depositAmount);
451. }
452. **if** (option == 2)
453. {
454. Console.Write("Enter Withdrawal Amount:\n");
455. **int** withdrawalAmount = **int**.Parse(Console.ReadLine());
456. **if** (withdrawalAmount <= accounts[i].withdrawalLimitProperty)
457. {
458. accounts[i].balanceProperty = (accounts[i].balanceProperty + withdrawalAmount);
459. accounts[i].withdraw(withdrawalAmount);
460. }
461. **else**
462. {
463. Console.Write("ERROR!!! Amount more than Withdrawal Limit.\n");
464. }
465. }
466. }
467. }
469. StreamWriter writeCurrent = **new** StreamWriter("CurrentAccount.txt"); //write to file object
471. **for** (**int** j = 0; j < totalAccounts; j++)
472. {
473. currentAccount account = accounts[j];
474. writeCurrent.WriteLine(account.accountNoProperty);
475. writeCurrent.WriteLine(account.accountTitleProperty);
476. writeCurrent.WriteLine(account.balanceProperty);
477. writeCurrent.WriteLine(account.cnicProperty);
478. writeCurrent.WriteLine(account.contactNoProperty);
479. writeCurrent.WriteLine(account.withdrawalLimitProperty);
480. }
481. writeCurrent.Close();
482. **return**;
483. }
485. **else** **if** (num == 2)
486. {
487. Console.WriteLine("\nEnter Account Number:");
488. **string** accnum = Console.ReadLine();
490. accountFile obj = **new** accountFile();
491. ArrayList accounts = obj.readAllSavingAccount(); //array list: saves diffeent types of objects
493. **int** totalAccounts = accounts.Count; //built-in count list function
494. **bool** found = **false**;
496. **for** (**int** i = 0; i < totalAccounts; i++)
497. {
498. **if** ((accounts[i] **as** savingAccount).accountNoProperty == accnum)
499. {
500. found = **true**;
501. Console.WriteLine("\nAccount Found.\n");
503. Console.Write("\nChoose Transaction Type:\n");
504. Console.Write("1.Deposit Amount.   2.Withdraw Amount.\n");
505. **int** option = **int**.Parse(Console.ReadLine());
507. **if** (option == 1)
508. {
509. Console.Write("Enter Deposit Amount:\n");
510. **int** depositAmount = **int**.Parse(Console.ReadLine());
511. (accounts[i] **as** savingAccount).balanceProperty = ((accounts[i] **as** savingAccount).balanceProperty + depositAmount);
512. (accounts[i] **as** savingAccount).deposit(depositAmount);
513. }
514. **if** (option == 2)
515. {
516. Console.Write("Enter Withdrawal Amount:\n");
517. **int** withdrawalAmount = **int**.Parse(Console.ReadLine());
518. (accounts[i] **as** savingAccount).balanceProperty = ((accounts[i] **as** savingAccount).balanceProperty + withdrawalAmount);
519. (accounts[i] **as** savingAccount).withdraw(withdrawalAmount);
520. }
521. }
522. }
524. StreamWriter writeSaving = **new** StreamWriter("SavingAccount.txt"); //write to file object
526. **for** (**int** j = 0; j < totalAccounts; j++)
527. {
528. savingAccount account = accounts[j] **as** savingAccount;
529. writeSaving.WriteLine(account.accountNoProperty);
530. writeSaving.WriteLine(account.accountTitleProperty);
531. writeSaving.WriteLine(account.balanceProperty);
532. writeSaving.WriteLine(account.cnicProperty);
533. writeSaving.WriteLine(account.contactNoProperty);
534. writeSaving.WriteLine(account.profitPercentageProperty);
535. }
536. writeSaving.Close();
537. **return**;
538. }
539. **else**
540. {
541. Console.WriteLine("Invalid Input\n");
542. }
543. }
544. }
545. }

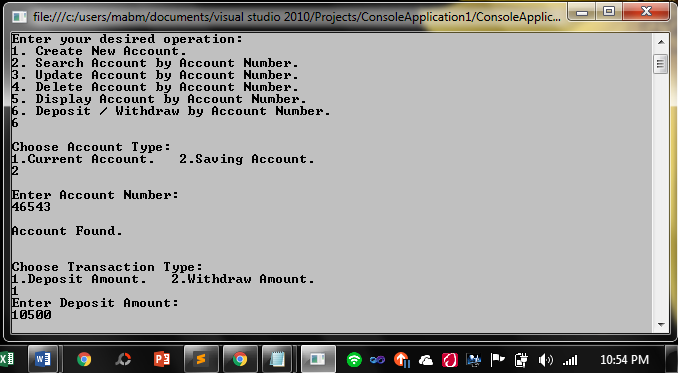
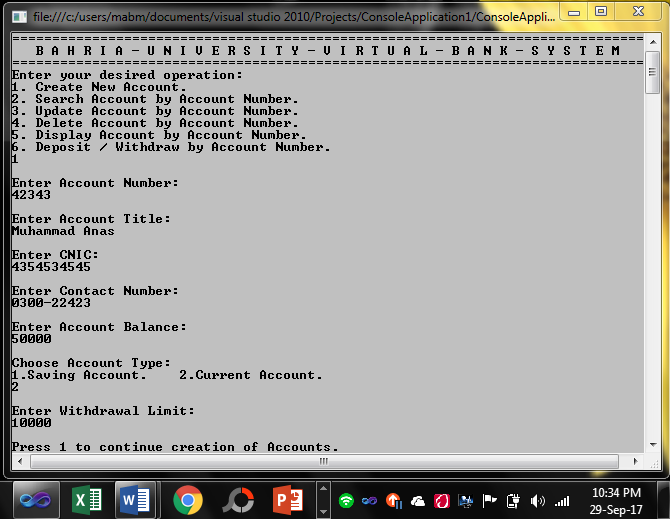
**accountManager.cs Class:**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Collections;
6. **using** System.IO;
8. **namespace** ConsoleApplication1
9. {
10. **class** accountManager
11. {
12. ArrayList savingAccountList = **new** ArrayList(); //array list: saves different types of objects
13. List<currentAccount> currentAccountList = **new** List<currentAccount>(); //simple list: saves same type of objects
15. **public** **void** createNewSavingAccount(savingAccount sa)
16. {
17. **this**.savingAccountList.Add(sa); //adding object to array list
18. }
20. **public** **void** createNewCurrentAccount(currentAccount ca)
21. {
22. **this**.currentAccountList.Add(ca); //adding object to simple list
23. }
25. **public** **void** writeAllinFile()
26. {
27. accountFile write = **new** accountFile();
28. write.writeAllSavingAccount(savingAccountList);
29. write.writeAllCurrentAccount(currentAccountList);
30. }
32. **public** **void** searchCurrentAccount(**string** An)
33. {
34. accountFile obj = **new** accountFile();
35. List<currentAccount> accounts = obj.readAllCurrentAccount(); //simple list: saves same type of objects
36. **int** totalAccounts = accounts.Count; //built-in count list function
37. **bool** found = **false**;
39. **for** (**int** i = 0; i < totalAccounts; i++)
40. {
41. **if** (accounts[i].accountNoProperty == An)
42. {
43. found = **true**;
44. Console.WriteLine("\nAccount Found.\n");
45. **return**;
46. }
47. }
48. Console.WriteLine("\nAccount Not Found.\n");
49. }
51. **public** **void** searchSavingAccount(**string** An)
52. {
53. accountFile obj = **new** accountFile();
54. ArrayList accounts = obj.readAllSavingAccount(); //array list: saves diffeent types of objects
55. **int** totalAccounts = accounts.Count; //built-in count list function
56. **bool** found = **false**;
58. **for** (**int** i = 0; i < totalAccounts; i++)
59. {
60. **if** ((accounts[i] **as** savingAccount).accountNoProperty == An)
61. {
62. found = **true**;
63. Console.WriteLine("\nAccount Found.\n");
64. **return**;
65. }
66. }
67. Console.WriteLine("\nAccount Not Found.\n");
68. }
69. }
70. }

**accountFile.cs Class:**

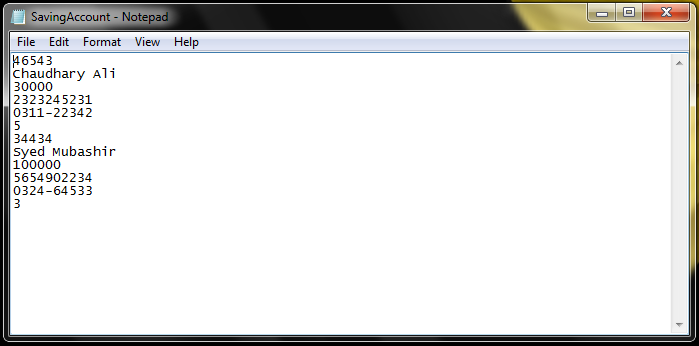
1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.IO;
6. **using** System.Collections;
8. **namespace** ConsoleApplication1
9. {
10. **class** accountFile
11. {
12. **public** **void** writeAllSavingAccount(ArrayList sa)
13. {
14. StreamWriter writeSaving = **new** StreamWriter("SavingAccount.txt"); //write to file object
15. **int** totalAccount = sa.Count;
17. **for** (**int** i = 0; i < totalAccount; i++)
18. {
19. savingAccount account = sa[i] **as** savingAccount;
20. writeSaving.WriteLine(account.accountNoProperty);
21. writeSaving.WriteLine(account.accountTitleProperty);
22. writeSaving.WriteLine(account.balanceProperty);
23. writeSaving.WriteLine(account.cnicProperty);
24. writeSaving.WriteLine(account.contactNoProperty);
25. writeSaving.WriteLine(account.profitPercentageProperty);
26. }
27. writeSaving.Close();
28. }
30. **public** **void** writeAllCurrentAccount(List<currentAccount> ca)
31. {
32. StreamWriter writeCurrent = **new** StreamWriter("CurrentAccount.txt"); //write to file object
33. **int** totalAccount = ca.Count;
35. **for** (**int** i = 0; i < totalAccount; i++)
36. {
37. currentAccount account = ca[i];
38. writeCurrent.WriteLine(account.accountNoProperty);
39. writeCurrent.WriteLine(account.accountTitleProperty);
40. writeCurrent.WriteLine(account.balanceProperty);
41. writeCurrent.WriteLine(account.cnicProperty);
42. writeCurrent.WriteLine(account.contactNoProperty);
43. writeCurrent.WriteLine(account.withdrawalLimitProperty);
44. }
45. writeCurrent.Close();
46. }
48. **public** ArrayList readAllSavingAccount()
49. {
50. StreamReader read = **new** StreamReader("SavingAccount.txt"); //read from file object
51. ArrayList savingAccount = **new** ArrayList();
52. savingAccount sv = **null**;
54. **while** (!read.EndOfStream)
55. {
56. **string** Ac = read.ReadLine();
57. **string** At = read.ReadLine();
58. **double** bal = **double**.Parse(read.ReadLine());
59. **string** cnic = read.ReadLine();
60. **string** cn = read.ReadLine();
61. **float** pp = **float**.Parse(read.ReadLine());
62. sv = **new** savingAccount(Ac, At, cnic, cn, bal, pp);
63. savingAccount.Add(sv);
64. }
65. read.Close();
66. **return** savingAccount;
67. }
69. **public** List<currentAccount> readAllCurrentAccount()
70. {
71. StreamReader read = **new** StreamReader("CurrentAccount.txt"); //read from file object
72. List<currentAccount> currentAccount = **new** List<currentAccount>();
73. currentAccount ca = **null**;
75. **while** (!read.EndOfStream)
76. {
77. **string** Ac = read.ReadLine();
78. **string** At = read.ReadLine();
79. **double** bal = **double**.Parse(read.ReadLine());
80. **string** cnic = read.ReadLine();
81. **string** cn = read.ReadLine();
82. **double** wl = **double**.Parse(read.ReadLine());
83. ca = **new** currentAccount(Ac, At, cnic, cn, bal, wl);
84. currentAccount.Add(ca);
85. }
86. read.Close();
87. **return** currentAccount;
88. }
89. }
90. }

**Console Output:**

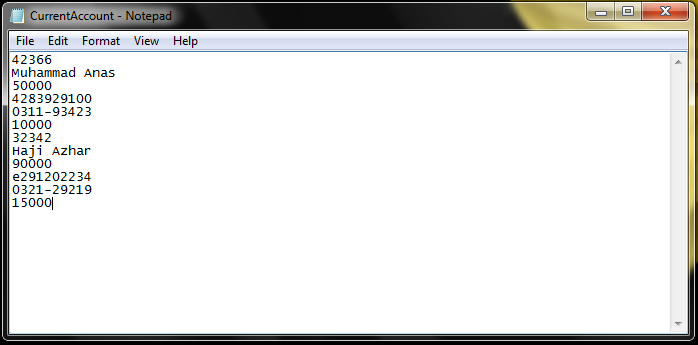
****

**Saved Files:**

**savingAccount.txt File:**

****

**currentAccount.txt File:**

****