

# DEAKIN UNIVERSITY

## OBJECT ORIENTED DEVELOPMENT

### ONTRACK SUBMISSION

---

# Bank Transactions

---

*Submitted By:*

Peter STACEY

pstacey

2020/04/27 14:23

*Tutor:*

Dipto PRATYAKSA

Outcome	Weight
Evaluate Code	◆◆◆◆◆
Principles	◆◆◆◆◆
Build Programs	◆◆◆◆◆
Design	◆◆◆◆◆
Justify	◆◆◆◆◆

This exercise significantly develops the banking application and contains quite a lot of new code and business rules. The task provides the basic guidance and design, however all of the implementation approach needs to be developed and coded, across multiple files. The task involves using many of the elements of object oriented development we have learnt to date, as well as a number of the features of C# such as exceptions, flow-control and iteration, that are relevant in program development.

April 27, 2020



```
1  using System;
2
3  namespace Task_4._2P
4  {
5      /// <summary>
6      /// Prototype for a Withdraw transaction
7      /// </summary>
8      class WithdrawTransaction
9      {
10         // Instance variables
11         private Account _account;
12         private decimal _amount;
13         private Boolean _executed;
14         private Boolean _success;
15         private Boolean _reversed;
16
17         // Properties
18         public Boolean Executed { get => _executed; }
19         public Boolean Success { get => _success; }
20         public Boolean Reversed { get => _reversed; }
21
22         /// <summary>
23         /// Constructs a WithdrawTransaction
24         /// </summary>
25         /// <param name="account">Account to withdraw from</param>
26         /// <param name="amount">Amount to withdraw</param>
27         public WithdrawTransaction(Account account, decimal amount)
28         {
29             _account = account;
30             if (amount > 0)
31             {
32                 _amount = amount;
33             }
34             else
35             {
36                 throw new ArgumentOutOfRangeException("Withdrawal amount must be >
37                     ↪ $0.00");
38             }
39             // _executed, _success, _reversed false by default
40         }
41
42         /// <summary>
43         /// Prints the details and status of the withdrawal
44         /// </summary>
45         public void Print()
46         {
47             Console.WriteLine(new String('-', 85));
48             Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|{3, 20}|",
49                 "ACCOUNT", "WITHDRAW AMOUNT", "STATUS", "CURRENT BALANCE");
50             Console.WriteLine(new String('-', 85));
51             Console.WriteLine("Withdrawal from {0, -20} | {1, 20} | {2, 20} | {3, 20} |",
52                 _account.Name,
53                 ↪ _amount.ToString("C"),
54                 _executed,
55                 _success,
56                 _reversed);
57         }
58     }
59 }
```

```
52         {
53             Console.Write("{0, 20}|" , "Pending");
54         }
55         else if (_reversed)
56         {
57             Console.Write("{0, 20}|" , "Withdraw reversed");
58         }
59         else if (_success)
60         {
61             Console.Write("{0, 20}|" , "Withdraw complete");
62         }
63         else if (!_success)
64         {
65             Console.Write("{0, 20}|" , "Insufficient funds");
66         }
67         Console.WriteLine("{0, 20}|" , _account.Balance.ToString("C"));
68         Console.WriteLine(new String('-', 85));
69     }
70
71     /// <summary>
72     /// Executes the withdrawal
73     /// </summary>
74     /// <exception cref="System.InvalidOperationException">Thrown
75     /// when the withdraw is already complete or insufficient funds</exception>
76     public void Execute()
77     {
78         if (_executed && _success)
79         {
80             throw new InvalidOperationException("Withdraw previously executed");
81         }
82         _executed = true;
83
84         _success = _account.Withdraw(_amount);
85         if (!_success)
86         {
87             throw new InvalidOperationException("Insufficient funds");
88         }
89     }
90
91     /// <summary>
92     /// Reverses the withdraw if previously executed successfully
93     /// </summary>
94     /// <exception cref="System.InvalidOperationException">Thrown
95     /// if already rolled back or if there are insufficient
96     /// funds to complete the rollback</exception>
97     public void Rollback()
98     {
99         if (_reversed)
100         {
101             throw new InvalidOperationException("Transaction already reversed");
102         }
103         else if (!_success)
104         {
```

```
105         throw new InvalidOperationException(  
106             "Withdraw not successfully executed. Nothing to rollback.");  
107     }  
108     _reversed = _account.Deposit(_amount); // Deposit returns boolean  
109     if (!_reversed) // Deposit didn't occur  
110     {  
111         throw new InvalidOperationException("Invalid amount");  
112     }  
113     _reversed = true;  
114 }  
115 }  
116 }
```

```
1  using System;
2
3  namespace Task_4._2P
4  {
5      /// <summary>
6      /// Prototype for a deposit transaction
7      /// </summary>
8      class DepositTransaction
9      {
10         // Instance variables
11         private Account _account;
12         private decimal _amount;
13         private Boolean _executed;
14         private Boolean _success;
15         private Boolean _reversed;
16
17         // Properties
18         public Boolean Executed { get => _executed; }
19         public Boolean Success { get => _success; }
20         public Boolean Reversed { get => _reversed; }
21
22         /// <summary>
23         /// Constructs a deposit transaction object
24         /// </summary>
25         /// <param name="account">Account to deposit into</param>
26         /// <param name="amount">Amount to deposit</param>
27         public DepositTransaction(Account account, decimal amount)
28         {
29             _account = account;
30             if (amount > 0)
31             {
32                 _amount = amount;
33             }
34             else
35             {
36                 throw new ArgumentOutOfRangeException(
37                     "Deposit amount invalid: {0}", amount.ToString("C"));
38             }
39             // _executed, _success, _reversed false by default
40         }
41
42         /// <summary>
43         /// Prints the details and status of a deposit
44         /// </summary>
45         public void Print()
46         {
47             Console.WriteLine(new String('-', 85));
48             Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|{3, 20}|",
49                 "ACCOUNT", "DEPOSIT AMOUNT", "STATUS", "CURRENT BALANCE");
50             Console.WriteLine(new String('-', 85));
51             Console.WriteLine("|{0, -20}|{1, 20}|", _account.Name,
52                 ↪ _amount.ToString("C"));
53             if (!_executed)
```

```
53         {
54             Console.Write("{0, 20}|" , "Pending");
55         }
56         else if (_reversed)
57         {
58             Console.Write("{0, 20}|" , "Deposit reversed");
59         }
60         else if (_success)
61         {
62             Console.Write("{0, 20}|" , "Deposit complete");
63         }
64         else if (!_success)
65         {
66             Console.Write("{0, 20}|" , "Invalid deposit");
67         }
68         Console.WriteLine("{0, 20}|" , _account.Balance.ToString("C"));
69         Console.WriteLine(new String('-', 85));
70     }
71
72     /// <summary>
73     /// Executes a deposit transaction
74     /// </summary>
75     public void Execute()
76     {
77         if (_executed && _success)
78         {
79             throw new InvalidOperationException("Deposit previously executed");
80         }
81         _executed = true;
82
83         _success = _account.Deposit(_amount);
84         if (!_success)
85         {
86             _executed = false;
87             throw new InvalidOperationException("Deposit amount invalid");
88         }
89     }
90
91     /// <summary>
92     /// Reverses a deposit if previously executed successfully
93     /// </summary>
94     public void Rollback()
95     {
96         if (_reversed)
97         {
98             throw new InvalidOperationException("Transaction already reversed");
99         }
100        else if (!_success)
101        {
102            throw new InvalidOperationException(
103                "Deposit not successfully executed. Nothing to rollback.");
104        }
105        _reversed = _account.Withdraw(_amount); // Withdraw returns boolean
```

```
106         if (!_reversed) // Withdraw didn't occur
107         {
108             throw new InvalidOperationException("Insufficient funds to
                ↳ rollback");
109         }
110         _reversed = true;
111     }
112 }
113 }
```

```
1  using System;
2
3  namespace Task_4._2P
4  {
5      /// <summary>
6      /// Prototype for a transfer transaction
7      /// </summary>
8      class TransferTransaction
9      {
10         // Instance variables
11         private Account _fromAccount;
12         private Account _toAccount;
13         private decimal _amount;
14         private DepositTransaction _deposit;
15         private WithdrawTransaction _withdraw;
16         private bool _executed;
17         private bool _reversed;
18
19         // Properties
20         public bool Executed { get => _executed; }
21         public bool Reversed { get => _reversed; }
22         public bool Success { get => (_deposit.Success && _withdraw.Success); }
23
24         /// <summary>
25         /// Constructor for a transfer transaction
26         /// </summary>
27         /// <param name="fromAccount">The account to transfer from</param>
28         /// <param name="toAccount">The account to transfer to</param>
29         /// <param name="amount">The amount to transfer</param>
30         /// <exception cref="System.ArgumentOutOfRangeException">Thrown
31         /// when the amount is negative</exception>
32         public TransferTransaction(Account fromAccount, Account toAccount, decimal
33         ↪ amount)
34         {
35             _fromAccount = fromAccount;
36             _toAccount = toAccount;
37             if (amount < 0)
38             {
39                 throw new ArgumentOutOfRangeException("Negative transfer amount");
40                 ↪ // THIS IS NOT GOOD. NEED TO ADJUST THIS
41             }
42             _amount = amount;
43
44             _withdraw = new WithdrawTransaction(_fromAccount, _amount);
45             _deposit = new DepositTransaction(_toAccount, _amount);
46         }
47
48         /// <summary>
49         /// Prints the details of the transfer
50         /// </summary>
51         public void Print()
52         {
53             Console.WriteLine(new String('-', 85));
```



```
52     Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|{3, 20}|",
53         "FROM ACCOUNT", "To ACCOUNT", "TRANSFER AMOUNT", "STATUS");
54     Console.WriteLine(new String('-', 85));
55     Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|", _fromAccount.Name,
56         ↪ _toAccount.Name, _amount.ToString("C"));
57     if (!_executed)
58     {
59         Console.WriteLine("{0, 20}|", "Pending");
60     }
61     else if (_reversed)
62     {
63         Console.WriteLine("{0, 20}|", "Transfer reversed");
64     }
65     else if (Success)
66     {
67         Console.WriteLine("{0, 20}|", "Transfer complete");
68     }
69     else if (!Success)
70     {
71         Console.WriteLine("{0, 20}|", "Transfer failed");
72     }
73     Console.WriteLine(new String('-', 85));
74 }
75
76 /// <summary>
77 /// Executes the transfer
78 /// </summary>
79 /// <exception cref="System.InvalidOperationException">Thrown
80 /// when previously executed or deposit or withdraw fail</exception>
81 public void Execute()
82 {
83     if (_executed)
84     {
85         throw new InvalidOperationException("Transfer previously executed");
86     }
87     _executed = true;
88
89     try
90     {
91         _withdraw.Execute();
92     }
93     catch (InvalidOperationException exception)
94     {
95         Console.WriteLine("Transfer failed with reason: " +
96             ↪ exception.Message);
97         _withdraw.Print();
98     }
99
100     if (_withdraw.Success)
101     {
102         try
103         {
104             _deposit.Execute();
105         }
106     }
107 }
```

```
103     }
104     catch (InvalidOperationException exception)
105     {
106         Console.WriteLine("Transfer failed with reason: " +
107             ↪ exception.Message);
108         _deposit.Print();
109         try
110         {
111             Rollback();
112         }
113         catch (InvalidOperationException e)
114         {
115             Console.WriteLine("Withdraw could not be reversed with
116                 ↪ reason: " + e.Message);
117             _withdraw.Print();
118         }
119     }
120     Print();
121     _deposit.Print();
122     _withdraw.Print();
123 }
124
125 /// <summary>
126 /// Rolls the transfer back
127 /// </summary>
128 /// <exception cref="System.InvalidOperationException">Thrown
129 /// when the rollback has already been executed or it fails</exception>
130 public void Rollback()
131 {
132     if (!_executed)
133     {
134         throw new InvalidOperationException("Transfer not executed. Nothing
135             ↪ to rollback.");
136     }
137
138     if (_reversed)
139     {
140         throw new InvalidOperationException("Transfer already rolled back");
141     }
142
143     if (this.Success)
144     {
145         try
146         {
147             _deposit.Rollback();
148         }
149         catch (InvalidOperationException exception)
150         {
151             Console.WriteLine("Failed to rollback deposit: "
152                 ↪ + exception.Message);
153             return;
154         }
155     }
156 }
```

```
153
154         try
155         {
156             _withdraw.Rollback();
157         }
158         catch (InvalidOperationException exception)
159         {
160             Console.WriteLine("Failed to rollback withdraw: "
161                             + exception.Message);
162             return;
163         }
164     }
165     _reversed = true;
166 }
167 }
168 }
```

```
1  using System;
2  using System.Diagnostics;
3
4  namespace Task_4._2P
5  {
6      enum MenuOption
7      {
8          Withdraw,
9          Deposit,
10         Transfer,
11         Print,
12         Quit
13     }
14
15     /// <summary>
16     /// BankSystem implements a banking system to operate on accounts
17     /// </summary>
18     class BankSystem
19     {
20         // Reads string input in the console
21         /// <summary>
22         /// Reads string input in the console
23         /// </summary>
24         /// <returns>
25         /// The string input of the user
26         /// </returns>
27         /// <param name="prompt">The string prompt for the user</param>
28         public static String ReadString(String prompt)
29         {
30             Console.Write(prompt + ": ");
31             return Console.ReadLine();
32         }
33
34         // Reads integer input in the console
35         /// <summary>
36         /// Reads integerinput in the console
37         /// </summary>
38         /// <returns>
39         /// The input of the user as an integer
40         /// </returns>
41         /// <param name="prompt">The string prompt for the user</param>
42         public static int ReadInteger(String prompt)
43         {
44             int number = 0;
45             string numberInput = ReadString(prompt);
46             while (!(int.TryParse(numberInput, out number)))
47             {
48                 Console.WriteLine("Please enter a whole number");
49                 numberInput = ReadString(prompt);
50             }
51             return Convert.ToInt32(numberInput);
52         }
53     }
```

```
54      // Reads integer input in the console between two numbers
55      /// <summary>
56      /// Reads integer input in the console between two numbers
57      /// </summary>
58      /// <returns>
59      /// The input of the user as an integer
60      /// </returns>
61      /// <param name="prompt">The string prompt for the user</param>
62      /// <param name="minimum">The minimum number allowed</param>
63      /// <param name="maximum">The maximum number allowed</param>
64      public static int ReadInteger(String prompt, int minimum, int maximum)
65      {
66          int number = ReadInteger(prompt);
67          while (number < minimum || number > maximum)
68          {
69              Console.WriteLine("Please enter a whole number from " +
70                              minimum + " to " + maximum);
71              number = ReadInteger(prompt);
72          }
73          return number;
74      }
75
76      // Reads decimal input in the console
77      /// <summary>
78      /// Reads decimal input in the console
79      /// </summary>
80      /// <returns>
81      /// The input of the user as a decimal
82      /// </returns>
83      /// <param name="prompt">The string prompt for the user</param>
84      public static decimal ReadDecimal(String prompt)
85      {
86          decimal number = 0;
87          string numberInput = ReadString(prompt);
88          while (!(decimal.TryParse(numberInput, out number)) || number <= 0)
89          {
90              Console.WriteLine("Please enter a decimal number greater than
91                               ↳ $0.00");
92              numberInput = ReadString(prompt);
93          }
94          return Convert.ToDecimal(numberInput);
95      }
96
97      /// <summary>
98      /// Displays a menu of possible actions for the user to choose
99      /// </summary>
100     private static void DisplayMenu()
101     {
102         Console.WriteLine("\n*****");
103         Console.WriteLine("*      Menu      *");
104         Console.WriteLine("*****");
105         Console.WriteLine("*  1. Withdraw  *");
106         Console.WriteLine("*  2. Deposit   *");
```

```
106         Console.WriteLine("* 3. Transfer    *");
107         Console.WriteLine("* 4. Print      *");
108         Console.WriteLine("* 5. Quit       *");
109         Console.WriteLine("*****");
110     }
111
112     /// <summary>
113     /// Returns a menu option chosen by the user
114     /// </summary>
115     /// <returns>
116     /// MenuOption chosen by the user
117     /// </returns>
118     static MenuOption ReadUserOption()
119     {
120         DisplayMenu();
121         int option = ReadInteger("Choose an option", 1,
122             Enum.GetNames(typeof(MenuOption)).Length);
123         return (MenuOption)(option - 1);
124     }
125
126     /// <summary>
127     /// Attempts to deposit funds into an account
128     /// </summary>
129     /// <param name="account">The account to deposit into</param>
130     static void DoDeposit(Account account)
131     {
132         decimal amount = ReadDecimal("Enter the amount");
133         DepositTransaction transaction = new DepositTransaction(account,
134             ↪ amount);
135         try
136         {
137             transaction.Execute();
138         }
139         catch (InvalidOperationException)
140         {
141             transaction.Print();
142             return;
143         }
144         transaction.Print();
145     }
146
147     /// <summary>
148     /// Attempts to withdraw funds from an account
149     /// </summary>
150     /// <param name="account">The account to withdraw from</param>
151     static void DoWithdraw(Account account)
152     {
153         decimal amount = ReadDecimal("Enter the amount");
154         WithdrawTransaction transaction = new WithdrawTransaction(account,
155             ↪ amount);
156         try
157         {
158             transaction.Execute();
159         }
160         catch (InvalidOperationException)
161         {
162             transaction.Print();
163             return;
164         }
165         transaction.Print();
166     }
167 }
```

```

157         }
158         catch (InvalidOperationException)
159         {
160             transaction.Print();
161             return;
162         }
163         transaction.Print();
164     }
165
166     /// <summary>
167     /// Attempts to transfer funds between accounts
168     /// </summary>
169     /// <param name="account">The account to withdraw from</param>
170     static void DoTransfer(Account fromAccount, Account toAccount) // this is
        ↳ temporary until we add multiple accounts in task 6.2
171     {
172         decimal amount = ReadDecimal("Enter the amount");
173         try
174         {
175             TransferTransaction transfer = new TransferTransaction(fromAccount,
        ↳ toAccount, amount);
176             transfer.Execute();
177         }
178         catch (Exception)
179         {
180             // Currently this is handled in the TransferTransaction. This will
        ↳ be changed
181         }
182     }
183
184     /// <summary>
185     /// Outputs the account name and balance
186     /// </summary>
187     /// <param name="account">The account to print</param>
188     static void DoPrint(Account account)
189     {
190         account.Print();
191     }
192
193     static void Main(string[] args)
194     {
195         /*****
196          *   TESTS
197          *****/
198         Account acc = new Account("Peter Stacey");
199         Account acc1 = new Account("Jane Doe", 100);
200         Account acc2 = new Account("John Doe", -500);
201
202         Debug.Assert(acc.Balance == 0);
203         Debug.Assert(acc1.Balance == 100);
204         Debug.Assert(acc2.Balance == 0);
205
206         // Test deposit success and rollback

```

```
207         DepositTransaction dep = new DepositTransaction(acc, 500);
208
209         dep.Print();
210         dep.Execute();
211         Debug.Assert(acc.Balance == 500);
212         Debug.Assert(dep.Executed == true);
213         Debug.Assert(dep.Success == true);
214         dep.Print();
215
216         dep.Rollback();
217         Debug.Assert(acc.Balance == 0);
218         Debug.Assert(dep.Reversed == true);
219         dep.Print();
220
221         Console.WriteLine("\n\n");
222
223         // Test withdraw success and rollback
224         WithdrawTransaction with = new WithdrawTransaction(acc1, 50);
225
226         with.Print();
227         with.Execute();
228         Debug.Assert(acc1.Balance == 50);
229         Debug.Assert(with.Executed == true);
230         Debug.Assert(with.Success == true);
231         with.Print();
232
233         with.Rollback();
234         Debug.Assert(acc1.Balance == 100);
235         Debug.Assert(with.Reversed == true);
236         with.Print();
237
238         Console.WriteLine("\n\n");
239
240         // Test transfer success and rollback
241         TransferTransaction tran = new TransferTransaction(acc1, acc, 50);
242
243         tran.Print();
244         tran.Execute();
245         Debug.Assert(acc.Balance == 50);
246         Debug.Assert(acc1.Balance == 50);
247         Debug.Assert(tran.Executed == true);
248         Debug.Assert(tran.Success == true);
249
250         tran.Rollback();
251         Debug.Assert(acc.Balance == 0);
252         Debug.Assert(acc1.Balance == 100);
253         Debug.Assert(tran.Reversed == true);
254         tran.Print();
255
256         Console.WriteLine("\n\n");
257
258         // Test withdraw failure when there is insufficient funds to complete  
→ the transaction
```



```
259      // followed by repeating the withdraw after funds are deposited
260      WithdrawTransaction with2 = new WithdrawTransaction(acc, 100);
261
262      with2.Print();
263      try
264      {
265          with2.Execute();
266      }
267      catch (InvalidOperationException exception)
268      {
269          Console.WriteLine(exception.Message);
270      }
271
272      Debug.Assert(acc.Balance == 0);
273      Debug.Assert(with2.Success == false);
274      Debug.Assert(with2.Executed == true);
275      with2.Print();
276
277      DepositTransaction dep2 = new DepositTransaction(acc, 500);
278
279      dep2.Execute();
280      dep2.Print();
281
282      try
283      {
284          with2.Execute();
285      }
286      catch (InvalidOperationException exception)
287      {
288          Console.WriteLine(exception.Message);
289      }
290
291      Debug.Assert(acc.Balance == 400);
292      Debug.Assert(with2.Success == true);
293      Debug.Assert(with2.Executed == true);
294      with2.Print();
295
296      Console.WriteLine("\n\n");
297
298      // Test fail to rollback before deposit or withdraw are
299      // complete
300      DepositTransaction dep3 = new DepositTransaction(acc, 500);
301      WithdrawTransaction with3 = new WithdrawTransaction(acc, 500);
302      TransferTransaction tran2 = new TransferTransaction(acc, acc1, 200);
303
304      try
305      {
306          dep3.Rollback();
307      }
308      catch (InvalidOperationException exception)
309      {
310          Console.WriteLine(exception.Message);
311      }
```

```
312
313     try
314     {
315         with3.Rollback();
316     }
317     catch (InvalidOperationException exception)
318     {
319         Console.WriteLine(exception.Message);
320     }
321
322     try
323     {
324         tran2.Rollback();
325     }
326     catch (InvalidOperationException exception)
327     {
328         Console.WriteLine(exception.Message);
329     }
330
331     Console.WriteLine("\n\n");
332
333     // Try to rollback deposit from account with insufficient funds
334     DepositTransaction dep4 = new DepositTransaction(acc2, 100);
335     WithdrawTransaction with4 = new WithdrawTransaction(acc2, 100);
336
337     dep4.Execute();
338     with4.Execute();
339     try
340     {
341         dep4.Rollback();
342     }
343     catch (Exception exception)
344     {
345         Console.WriteLine(exception.Message);
346     }
347
348     Console.WriteLine("\n\n");
349
350     /* *****
351      * CLI
352      * ***** */
353     do
354     {
355         MenuOption chosen = ReadUserOption();
356         switch (chosen)
357         {
358             case MenuOption.Withdraw:
359                 DoWithdraw(acc); break;
360             case MenuOption.Deposit:
361                 DoDeposit(acc); break;
362             case MenuOption.Transfer:
363                 DoTransfer(acc, acc1); break;
364             case MenuOption.Print:
```

```
365         DoPrint(acc); break;
366     case MenuOption.Quit:
367     default:
368         Console.WriteLine("Goodbye");
369         System.Environment.Exit(0); // terminates the program
370         break; // unreachable
371     }
372 } while (true);
373 }
374 }
375 }
```

```
1  using System;
2
3  namespace Task_4._2P
4  {
5      /// <summary>
6      /// A bank account class to hold the account name and balance details
7      /// </summary>
8      class Account
9      {
10         // Instance variables
11         private String _name;
12         private decimal _balance;
13
14         // Read-only properties
15         public String Name { get => _name; }
16         public decimal Balance { get => _balance; }
17
18
19         /// <summary>
20         /// Class constructor
21         /// </summary>
22         /// <param name="name">The name string for the account</param>
23         /// <param name="balance">The decimal balance of the account</param>
24         public Account(String name, decimal balance = 0)
25         {
26             _name = name;
27             if (balance <= 0)
28                 return;
29             _balance = balance;
30         }
31
32         /// <summary>
33         /// Deposits money into the account
34         /// </summary>
35         /// <returns>
36         /// Boolean whether the deposit was successful (true) or not (false)
37         /// </returns>
38         /// <param name="amount">The decimal amount to add to the balance</param>
39         public Boolean Deposit(decimal amount)
40         {
41             if ((amount < 0) || (amount == decimal.MaxValue))
42                 return false;
43
44             _balance += amount;
45             return true;
46         }
47
48         /// <summary>
49         /// Withdraws money from the account (with no overdraw protection currently)
50         /// </summary>
51         /// <returns>
52         /// Boolean whether the withdrawal was successful (true) or not (false)
53         /// </returns>
```

```
54     /// <param name="amount">The amount to subtract from the balance</param>
55     public Boolean Withdraw(decimal amount)
56     {
57         if ((amount < 0) || (amount > _balance))
58             return false;
59
60         _balance -= amount;
61         return true;
62     }
63
64     /// <summary>
65     /// Outputs the account name and current balance as a string
66     /// </summary>
67     public void Print()
68     {
69         Console.WriteLine("Account Name: {0}, Balance: {1}",
70             _name, _balance.ToString("C"));
71     }
72 }
73 }
```