

DEAKIN UNIVERSITY

OBJECT ORIENTED DEVELOPMENT

ONTRACK SUBMISSION

Multiple Bank Accounts

Submitted By:

Peter STACEY

pstacey

2020/05/01 10:56

Tutor:

Dipto PRATYAKSA

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

As a continuation of the banking application, overall the app is growing into a larger design and work of coding, however this individual task, while important, was relatively small. It involved evaluating our existing code in order to extend it with new features, and the addition of a new class, which involves new code writing. My video will include additional evidence to align with the learning outcomes.

May 1, 2020



```
1  using System;
2  using System.Diagnostics;
3
4  namespace Task_6._2P
5  {
6      enum MenuOption
7      {
8          CreateAccount,
9          Withdraw,
10         Deposit,
11         Transfer,
12         Print,
13         Quit
14     }
15
16     /// <summary>
17     /// BankSystem implements a banking system to operate on accounts
18     /// </summary>
19     class BankSystem
20     {
21         // Reads string input in the console
22         /// <summary>
23         /// Reads string input in the console
24         /// </summary>
25         /// <returns>
26         /// The string input of the user
27         /// </returns>
28         /// <param name="prompt">The string prompt for the user</param>
29         public static String ReadString(String prompt)
30         {
31             Console.Write(prompt + ": ");
32             return Console.ReadLine();
33         }
34
35         // Reads integer input in the console
36         /// <summary>
37         /// Reads integerinput in the console
38         /// </summary>
39         /// <returns>
40         /// The input of the user as an integer
41         /// </returns>
42         /// <param name="prompt">The string prompt for the user</param>
43         public static int ReadInteger(String prompt)
44         {
45             int number = 0;
46             string numberInput = ReadString(prompt);
47             while (!(int.TryParse(numberInput, out number)))
48             {
49                 Console.WriteLine("Please enter a whole number");
50                 numberInput = ReadString(prompt);
51             }
52             return Convert.ToInt32(numberInput);
53         }
54     }
55 }
```

```
54
55 // Reads integer input in the console between two numbers
56 /// <summary>
57 /// Reads integer input in the console between two numbers
58 /// </summary>
59 /// <returns>
60 /// The input of the user as an integer
61 /// </returns>
62 /// <param name="prompt">The string prompt for the user</param>
63 /// <param name="minimum">The minimum number allowed</param>
64 /// <param name="maximum">The maximum number allowed</param>
65 public static int ReadInteger(String prompt, int minimum, int maximum)
66 {
67     int number = ReadInteger(prompt);
68     while (number < minimum || number > maximum)
69     {
70         Console.WriteLine("Please enter a whole number from " +
71             minimum + " to " + maximum);
72         number = ReadInteger(prompt);
73     }
74     return number;
75 }
76
77 // Reads decimal input in the console
78 /// <summary>
79 /// Reads decimal input in the console
80 /// </summary>
81 /// <returns>
82 /// The input of the user as a decimal
83 /// </returns>
84 /// <param name="prompt">The string prompt for the user</param>
85 public static decimal ReadDecimal(String prompt)
86 {
87     decimal number = 0;
88     string numberInput = ReadString(prompt);
89     while (!(decimal.TryParse(numberInput, out number)) || number < 0)
90     {
91         Console.WriteLine("Please enter a decimal number, $0.00 or
92             ↵ greater");
93         numberInput = ReadString(prompt);
94     }
95     return Convert.ToDecimal(numberInput);
96 }
97
98 /// <summary>
99 /// Displays a menu of possible actions for the user to choose
100 /// </summary>
101 private static void DisplayMenu()
102 {
103     Console.WriteLine("\n*****");
104     Console.WriteLine("*      Menu      *");
105     Console.WriteLine("*****");
106     Console.WriteLine("*  1. New Account  *");
107 }
```

```
106         Console.WriteLine("* 2. Withdraw    *");
107         Console.WriteLine("* 3. Deposit     *");
108         Console.WriteLine("* 4. Transfer    *");
109         Console.WriteLine("* 5. Print      *");
110         Console.WriteLine("* 6. Quit       *");
111         Console.WriteLine("*****");
112     }
113
114     /// <summary>
115     /// Returns a menu option chosen by the user
116     /// </summary>
117     /// <returns>
118     /// MenuOption chosen by the user
119     /// </returns>
120     static MenuOption ReadUserOption()
121     {
122         DisplayMenu();
123         int option = ReadInteger("Choose an option", 1,
124             Enum.GetNames(typeof(MenuOption)).Length);
125         return (MenuOption)(option - 1);
126     }
127
128     /// <summary>
129     /// Attempts to deposit funds into an account at a bank
130     /// </summary>
131     /// <param name="bank">The bank holding the account to deposit into</param>
132     static void DoDeposit(Bank bank)
133     {
134         Account account = FindAccount(bank);
135         if (account != null)
136         {
137             decimal amount = ReadDecimal("Enter the amount");
138             DepositTransaction transaction = new DepositTransaction(account,
139                 ↪ amount);
139             try
140             {
141                 bank.ExecuteTransaction(transaction);
142             }
143             catch (InvalidOperationException)
144             {
145                 transaction.Print();
146                 return;
147             }
148             transaction.Print();
149         }
150     }
151
152     /// <summary>
153     /// Attempts to withdraw funds from an account at a bank
154     /// </summary>
155     /// <param name="bank">The bank holding account to withdraw from</param>
156     static void DoWithdraw(Bank bank)
157     {
```

```
158     Account account = FindAccount(bank);
159     if (account != null)
160     {
161         decimal amount = ReadDecimal("Enter the amount");
162         WithdrawTransaction transaction = new WithdrawTransaction(account,
163             ↪ amount);
164         try
165         {
166             bank.ExecuteTransaction(transaction);
167         }
168         catch (InvalidOperationException)
169         {
170             transaction.Print();
171             return;
172         }
173         transaction.Print();
174     }
175
176     /// <summary>
177     /// Attempts to transfer funds between accounts
178     /// </summary>
179     /// <param name="bank">The bank holding the account
180     /// to transfer between</param>
181     static void DoTransfer(Bank bank)
182     {
183         Console.WriteLine("Transfer from:");
184         Account from = FindAccount(bank);
185         Console.WriteLine("Transfer to:");
186         Account to = FindAccount(bank);
187         if (from != null && to != null)
188         {
189             decimal amount = ReadDecimal("Enter the amount");
190             try
191             {
192                 TransferTransaction transaction = new TransferTransaction(from,
193                     ↪ to, amount);
194                 bank.ExecuteTransaction(transaction);
195                 transaction.Print();
196             }
197             catch (Exception)
198             {
199                 // Currently this is handled in the TransferTransaction. This
200                 ↪ will be changed
201             }
202         }
203     }
204
205     /// <summary>
206     /// Outputs the account name and balance
207     /// </summary>
208     /// <param name="account">The account to print</param>
209     static void DoPrint(Bank bank)
```

```
208     {
209         Account account = FindAccount(bank);
210         if (account != null)
211         {
212             account.Print();
213         }
214     }
215
216     /// <summary>
217     /// Creates a new account and adds it to the Bank
218     /// </summary>
219     /// <param name="bank">The bank to create the account in</param>
220     static void CreateAccount(Bank bank)
221     {
222         string name = ReadString("Enter account name");
223         decimal balance = ReadDecimal("Enter the opening balance");
224         bank.AddAccount(new Account(name, balance));
225     }
226
227     private static Account FindAccount(Bank bank)
228     {
229         Account account = null;
230         string name = ReadString("Enter the account name");
231         account = bank.GetAccount(name);
232         if (account == null)
233         {
234             Console.WriteLine("That account name does not exist at this bank");
235         }
236         return account;
237     }
238
239     static void Main(string[] args)
240     {
241         Bank bank = new Bank();
242
243         do
244         {
245             MenuOption chosen = ReadUserOption();
246             switch (chosen)
247             {
248                 case MenuOption.CreateAccount:
249                     CreateAccount(bank); break;
250
251                 case MenuOption.Withdraw:
252                     DoWithdraw(bank); break;
253
254                 case MenuOption.Deposit:
255                     DoDeposit(bank); break;
256
257                 case MenuOption.Transfer:
258                     DoTransfer(bank); break;
259
260                 case MenuOption.Print:
```

```
261             DoPrint(bank); break;
262
263         case MenuOption.Quit:
264         default:
265             Console.WriteLine("Goodbye");
266             System.Environment.Exit(0); // terminates the program
267             break; // unreachable
268     }
269 } while (true);
270 }
271 }
272 }
```

```
1  using System;
2  using System.Collections.Generic;
3
4  namespace Task_6._2P
5  {
6      /// <summary>
7      /// Prototype for a bank to hold accounts
8      /// </summary>
9      class Bank
10     {
11         // Instance variables
12         private List<Account> _accounts;
13
14         /// <summary>
15         /// Creates an empty bank object with a list for accounts
16         /// </summary>
17         public Bank()
18         {
19             _accounts = new List<Account>();
20         }
21
22         /// <summary>
23         /// Adds an account to the Bank accounts register
24         /// </summary>
25         /// <param name="account"></param>
26         public void AddAccount(Account account)
27         {
28             _accounts.Add(account);
29         }
30
31         /// <summary>
32         /// Returns the first Account corresponding to the name, or
33         /// null if there is no account matching the criteria
34         /// </summary>
35         /// <param name="name"></param>
36         /// <returns>
37         /// Account matching the provided name, or null
38         /// </returns>
39         public Account GetAccount(string name)
40         {
41             foreach (Account account in _accounts)
42             {
43                 if (account.Name == name)
44                 {
45                     return account;
46                 }
47             }
48             return null;
49         }
50
51         /// <summary>
52         /// Executes a deposit into an account
53         /// </summary>
```



```
54     /// <param name="transaction">DepositTransaction to execute</param>
55     public void ExecuteTransaction(DepositTransaction transaction)
56     {
57         try
58         {
59             transaction.Execute();
60         }
61         catch (InvalidOperationException exception)
62         {
63             Console.WriteLine("An error occurred in executing the transaction");
64             Console.WriteLine("The error was: " + exception.Message);
65         }
66     }
67
68     /// <summary>
69     /// Executes a WithdrawTransaction on an account
70     /// </summary>
71     /// <param name="transaction">WithdrawTransaction to execute</param>
72     public void ExecuteTransaction(WithdrawTransaction transaction)
73     {
74         try
75         {
76             transaction.Execute();
77         }
78         catch (InvalidOperationException exception)
79         {
80             Console.WriteLine("An error occurred in executing the transaction");
81             Console.WriteLine("The error was: " + exception.Message);
82         }
83     }
84
85     /// <summary>
86     /// Transfers funds between accounts held by the bank
87     /// </summary>
88     /// <param name="transaction">TransferTransaction to execute</param>
89     public void ExecuteTransaction(TransferTransaction transaction)
90     {
91         try
92         {
93             transaction.Execute();
94         }
95         catch (InvalidOperationException exception)
96         {
97             Console.WriteLine("An error occurred in executing the transaction");
98             Console.WriteLine("The error was: " + exception.Message);
99         }
100     }
101 }
102 }
```

```
1  using System;
2
3  namespace Task_6._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 }
```

```
1  using System;
2
3  namespace Task_6._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 Details: {0, -20}|{1, 20}|", _account.Name,
52                 ↪ _amount.ToString("C"));
53             if (!_executed)
```

```
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_6._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_6._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             }
41             _amount = amount;
42
43             _withdraw = new WithdrawTransaction(_fromAccount, _amount);
44             _deposit = new DepositTransaction(_toAccount, _amount);
45         }
46
47         /// <summary>
48         /// Prints the details of the transfer
49         /// </summary>
50         public void Print()
51         {
52             Console.WriteLine(new String('-', 85));
53             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 }
```

```
104         catch (InvalidOperationException exception)
105         {
106             Console.WriteLine("Transfer failed with reason: " +
107                 ↪ exception.Message);
108             _deposit.Print();
109             try
110             {
111                 _withdraw.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     }
121
122     /// <summary>
123     /// Rolls the transfer back
124     /// </summary>
125     /// <exception cref="System.InvalidOperationException">Thrown
126     /// when the rollback has already been executed or it fails</exception>
127     public void Rollback()
128     {
129         if (!_executed)
130         {
131             throw new InvalidOperationException("Transfer not executed. Nothing
132                 ↪ to rollback.");
133         }
134
135         if (_reversed)
136         {
137             throw new InvalidOperationException("Transfer already rolled back");
138         }
139
140         if (this.Success)
141         {
142             try
143             {
144                 _deposit.Rollback();
145             }
146             catch (InvalidOperationException exception)
147             {
148                 Console.WriteLine("Failed to rollback deposit: "
149                     ↪ + exception.Message);
150                 return;
151             }
152
153             try
154             {
155                 _withdraw.Rollback();
156             }
157             catch (InvalidOperationException exception)
158             {
159                 Console.WriteLine("Failed to rollback withdraw: "
160                     ↪ + exception.Message);
161                 return;
162             }
163         }
164     }
165 }
```

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