

# DEAKIN UNIVERSITY

## OBJECT ORIENTED DEVELOPMENT

### ONTRACK SUBMISSION

---

# Implementing Abstract Transactions

---

*Submitted By:*

Connor GENT

gentco

2021/05/22 20:17

*Tutor:*

Nayyar ZAIDI

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

Here is my 7.1 task which focused on reducing the level of code duplication.

May 22, 2021



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Connor_Gent_7._1_task
8  {
9      class DepositTransaction: Transaction
10     {
11         private Account _account;
12
13
14
15
16         public DepositTransaction (Account account, decimal amount) : base(amount)
17
18         {
19
20             this._account = account;
21
22
23
24         }
25
26
27         public override void Print()
28         {
29             Console.WriteLine("Transaction Successful: " + Executed + "\nDeposited
30                 ↳ " + _amount + " To " + _account.Name + ". Trasaction exected at "
31                 ↳ + DateStamp);
32         }
33
34         public override void Rollback()
35         {
36             base.Rollback();
37
38             try
39             {
40                 if(Success == false)
41                 {
42                     throw new InvalidOperationException("Transaction was not
43                         ↳ succesful");
44                 }
45                 if (Reversed)
46                 {
47                     throw new InvalidOperationException("Transaction again was not
48                         ↳ successful");
49                 }
50                 else
51                 {
52                     _account.Withdraw(_amount);
53                     _reversed = true;
54                 }
55             }
56         }
57     }
58 }
```

```
50         }
51     }
52     catch (InvalidOperationException exception)
53     {
54         Console.WriteLine("There was an error detected: " +
55             ↳ exception.GetType().ToString() + "With message \"" +
56             ↳ exception.Message + "\"");
57     }
58
59     public override void Execute()
60     {
61         base.Execute();
62         try
63         {
64             if(_amount < 0 )
65             {
66                 throw new InvalidOperationException();
67             }
68             if (Executed)
69             {
70                 throw new InvalidOperationException();
71             }
72             else
73             {
74                 _account.Deposit(_amount);
75                 _account.Deposit(_amount);
76                 _success = true;
77                 _excuted = true;
78                 _datestamp = DateTime.Now;
79                 Print();
80             }
81         }
82     }
83
84     catch (InvalidOperationException)
85     {
86         Console.WriteLine(" Error with account. Transaction could not take
87             ↳ place. ");
88     }
89 }
90
91
92
93
94
95
96
97
98
99
```

100  
101  
102  
103       }  
104    }

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Connor_Gent_7._1_task
8  {
9      class WithdrawTransaction : Transaction
10     {
11         private Account _account;
12
13
14
15
16         public WithdrawTransaction(Account account, decimal _amount) : base
17             ↳ (_amount)
18         {
19
20             this._account = account;
21
22
23
24
25
26
27         }
28
29
30         public override void Print()
31         {
32
33             Console.WriteLine("Transaction successful: " + Executed + "\nWithdrawn:
34             ↳ " + _amount
35               + " from " + _account.Name + ". Transaction executed at " +
36             ↳ DateStamp);
37
38         }
39
40         public override void Rollback()
41         {
42             base.Rollback();
43
44             try
45             {
46                 if (Success == false)
47                 {
48                     throw new InvalidOperationException("Transaction was not
49                     ↳ successful");
50                 }
51                 if (_reversed)
```

```
50         {
51             throw new InvalidOperationException("Transaction again was
52             ↪ not successful");
53         }
54     else
55     {
56         _account.Balance += _amount;
57         _reversed = true;
58     }
59 }
60
61 catch (InvalidOperationException)
62 {
63     Console.WriteLine("Transaction could not go forward. Check account
64     ↪ to fix " + GetType().ToString());
65 }
66
67
68 public override void Execute()
69 {
70     base.Execute();
71
72     try
73     {
74
75
76         if (_amount > _account.Balance)
77         {
78
79             throw new InvalidOperationException("Insufficient funds");
80         }
81
82         if (Executed)
83         {
84
85             throw new InvalidOperationException("Transaction already
86             ↪ attempted");
87         }
88         if (_amount < 0)
89         {
90             throw new InvalidOperationException("Please enter a valid
91             ↪ amount");
92         }
93     else
94     {
95         _account.Withdraw(_amount);
96         _success = true;
97         _excuted = true;
98         _datestamp = DateTime.Now;
```

```
99         Print();
100     }
101 }
102 catch (InvalidOperationException)
103 {
104     Console.WriteLine("Transaction could not go forward. Check account
        ↳ to fix " + GetType().ToString());
105 }
106
107 }
108
109 }
110 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Connor_Gent_7._1_task
8  {
9      class TransferTransaction : Transaction
10     {
11         private Account _toaccount, _fromaccount;
12
13         private DepositTransaction _deposit;
14
15         private WithdrawTransaction _withdraw;
16
17
18
19
20
21         public TransferTransaction(Account toaccount, Account fromaccount, decimal
            ↳ _amount) : base(_amount) //constructor
22         {
23
24
25             this._toaccount = toaccount;
26
27             this._fromaccount = fromaccount;
28
29             this._amount = _amount;
30
31
32
33         }
34
35         public override void Print()
36         {
37
38
39             Console.WriteLine("Transaction Successful: " + Executed +
            ↳ "\nTransferred: " + _amount
40             + " from " + _fromaccount.Name + " To " + _toaccount.Name +
            ↳ DateStamp);
41         }
42
43         public override void Rollback()
44         {
45             base.Rollback();
46
47
48             try
49
50             {
```



```
51         if (Success == false)
52         {
53             throw new InvalidOperationException("Transaction was not
54                 ↪ successful");
55         }
56         if (Reversed)
57         {
58             throw new InvalidOperationException("Transaction again was not
59                 ↪ successful");
60         }
61         if (_amount > _toaccount.Balance)
62         {
63             throw new InvalidOperationException("The " + _toaccount.Name +
64                 ↪ "Account does not have enough money");
65         }
66         else
67         {
68             _deposit.Rollback();
69             _withdraw.Rollback();
70             _reversed = true;
71         }
72     }
73
74     catch (InvalidOperationException)
75     {
76         Console.WriteLine("Transaction could not be completed with amount
77             ↪ of funds in the account " + GetType().ToString());
78     }
79
80 }
81
82 public override void Execute()
83 {
84     base.Execute();
85
86     try
87     {
88         if (_amount > _fromaccount.Balance)
89         {
90             throw new InvalidOperationException("Insufficient funds");
91         }
92     }
93     if (Executed)
94     {
95         throw new InvalidOperationException("Transaction already
96             ↪ attempted");
97     }
98 }
```

```
99         else
100         {
101             _withdraw = new WithdrawTransaction(_fromaccount, _amount);
102             _withdraw.Execute();
103
104             if (_withdraw.Success)
105             {
106                 _deposit = new DepositTransaction(_toaccount, _amount);
107                 _deposit.Execute();
108
109                 if (_deposit.Success)
110                 {
111                     _success = true;
112                     _excuted = true;
113                     _datestamp = DateTime.Now;
114
115
116                     Print();
117                 }
118                 else
119                 {
120                     Rollback();
121                 }
122             }
123         }
124     }
125     catch (InvalidOperationException)
126     {
127         Console.WriteLine("Transaction could not be completed with amount
128             ↳ of funds in the account");
129     }
130 }
131 }
132
133
134
135
136
137
```

```
1  using System;
2
3  namespace Connor_Gent_7._1_task
4  {
5      abstract class Transaction
6      {
7          protected decimal _amount;
8          protected bool _success;
9          public bool _excuted, _reversed;
10         public DateTime _datestamp;
11
12         public bool Success { get => _success; }
13
14         public bool Executed { get => _excuted; }
15
16         public bool Reversed { get => _reversed; }
17
18         public DateTime DateStamp { get => _datestamp; }
19
20         public decimal Amount { get => _amount; }
21
22         public Transaction(decimal amount)
23         {
24             this._amount = amount;
25         }
26
27         abstract public void Print();
28
29         public virtual void Execute()
30         {
31
32         }
33
34         public virtual void Rollback()
35         {
36             if (_reversed)
37             {
38                 throw new InvalidOperationException("Transaction already reversed");
39             }
40             else if (!_success)
41             {
42                 throw new InvalidOperationException("Transaction not successful.
43                     ↪ Nothing to Rollback");
44             }
45
46             _datestamp = DateTime.Now;
47         }
48     }
49
50
51
52 }
```

53 }

```
1  using System;
2
3  namespace Connor_Gent_7._1_task
4  {
5      public enum MenuOption
6      {
7          Withdraw = 1,
8
9          Deposit = 2,
10
11         Transfer = 3,
12
13         AddAccount = 4,
14
15         FindAccount = 5,
16
17         PrintTransactionHistory = 6,
18
19         Print = 7,
20
21         Quit = 8
22     }
23
24     class BankSystem
25     {
26         static MenuOption ReadUserOption()
27         {
28             int choice = 0;
29
30             do
31             {
32                 Console.WriteLine("1. Withdraw");
33
34                 Console.WriteLine("2. Deposit");
35
36                 Console.WriteLine("3. Transfer");
37
38                 Console.WriteLine("4. Add new account");
39
40                 Console.WriteLine("5. Find Account");
41
42                 Console.WriteLine("6. Print transaction account");
43
44                 Console.WriteLine("7. Print");
45
46                 Console.WriteLine("8. Quit");
47
48             } while (choice != 0);
49
50             return choice;
51         }
52     }
53 }
```

```
54         Console.WriteLine("Enter choice: ");
55
56         try
57         {
58
59             choice = Convert.ToInt32(Console.ReadLine());
60
61         }
62
63         catch (Exception) { }
64
65     } while (choice < 1 || choice > 8);
66
67     return (MenuOption)choice;
68
69 }
70
71
72 public static Bank bank = new Bank();
73
74 static void Main(string[] args)
75
76 {
77
78
79
80     Account Jason = new Account(420, "Jason");
81
82     bank.AddAccount(Jason);
83
84     Jason.Deposit(200);
85
86     Jason.Withdraw(500);
87
88     Jason.Print();
89
90
91     Account James = new Account(420, "James");
92
93     bank.AddAccount(James);
94
95     James.Deposit(300);
96
97     James.Withdraw(40);
98
99     James.Print();
100
101     while (true)
102     {
103
104
105         switch (ReadUserOption())
106
```

```
107         {
108
109             case MenuOption.Withdraw:
110
111                 DoWithdraw(bank);
112
113                 break;
114
115             case MenuOption.Deposit:
116
117                 DoDeposit(bank);
118
119                 break;
120
121             case MenuOption.Transfer:
122
123                 DoTransfer(bank);
124
125                 break;
126
127             case MenuOption.AddAccount:
128
129                 bank.AddAccount(GetAccount());
130
131                 break;
132             case MenuOption.FindAccount:
133                 FindAccount(bank);
134                 break;
135             case MenuOption.PrintTransactionHistory:
136                 DoPrintTransactionHistory(bank);
137                 break;
138
139             case MenuOption.Print:
140
141                 DoPrint(bank);
142
143                 break;
144
145             case MenuOption.Quit:
146
147                 Environment.Exit(0);
148
149                 break;
150
151             default:
152                 Jason.Quit();
153                 break;
154
155
156         }
157
158     }
159 }
```

```
160
161     }
162
163     static Account GetAccount()
164
165     {
166
167         Console.WriteLine("Enter account name: ");
168
169         String name = Console.ReadLine();
170
171         Console.WriteLine("Enter starting balance: ");
172
173         decimal balance = Convert.ToDecimal(Console.ReadLine());
174
175         return new Account(balance, name);
176
177     }
178
179     static Account FindAccount(Bank bank)
180
181     {
182
183         Console.WriteLine("Enter account name: ");
184
185         string name = Console.ReadLine();
186
187         var account = bank.GetAccount(name);
188
189         if (account == null)
190
191         {
192
193             Console.WriteLine("Account wiht name " + name + " not found");
194
195         }
196
197         return account;
198
199     }
200
201     static void DoWithdraw(Bank bank)
202
203     {
204
205         var account = FindAccount(bank);
206
207         if (account == null)
208
209             return;
210
211         Console.WriteLine("Enter value: ");
212
```



```
213         decimal amount = Convert.ToDecimal(Console.ReadLine());
214
215         WithdrawTransaction withdrawTransaction = new
216             ↳ WithdrawTransaction(account, amount);
217
218         bank.ExecuteTransaction(withdrawTransaction);
219     }
220
221     static void DoDeposit(Bank bank)
222     {
223
224         var account = FindAccount(bank);
225
226         if (account == null)
227
228             return;
229
230         Console.WriteLine("Enter value: ");
231
232         decimal amount = Convert.ToDecimal(Console.ReadLine());
233
234         DepositTransaction depositTransaction = new DepositTransaction(account,
235             ↳ amount);
236
237         bank.ExecuteTransaction(depositTransaction);
238     }
239
240     static void DoPrintTransactionHistory(Bank bank)
241     {
242         bank.PrintTransactionHistory();
243     }
244
245     static void DoTransfer(Bank bank)
246     {
247
248         Console.WriteLine("From Account:");
249
250         var fromAccount = FindAccount(bank);
251
252         if (fromAccount == null)
253
254             return;
255
256         Console.WriteLine("To Account:");
257
258         var toAccount = FindAccount(bank);
259
260         if (toAccount == null)
```

```
264         {
265
266             return;
267
268         }
269
270         Console.WriteLine("Enter amount: ");
271
272         decimal amount = Convert.ToDecimal(Console.ReadLine());
273
274         TransferTransaction transferTransaction = new
275             ↪ TransferTransaction(toAccount, fromAccount, amount);
276
277         bank.ExecuteTransaction(transferTransaction);
278     }
279
280     static void DoPrint(Bank bank)
281
282     {
283
284         var account = FindAccount(bank);
285
286         if (account != null)
287
288         {
289
290             account.Print();
291
292         }
293
294         else
295
296         {
297
298             Console.WriteLine("Account not found");
299
300         }
301
302
303
304     }
305
306     public static void DoRollBack(Transaction transaction)
307     {
308         transaction.Rollback();
309     }
310 }
311 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Connor_Gent_7._1_task
8  {
9      class Bank
10     {
11         private List<Account> accountList;
12
13
14         private List<Transaction> _transactions;
15
16         public List<Transaction> Transactions { get => _transactions; }
17
18
19
20         public Bank()
21         {
22
23             accountList = new List<Account>();
24             _transactions = new List<Transaction>();
25
26         }
27
28
29
30
31         public void AddAccount(Account account)
32         {
33
34             accountList.Add(account);
35
36         }
37
38
39
40
41         public Account GetAccount(String name)
42         {
43
44             return accountList.FirstOrDefault(a => a.Name == name);
45
46         }
47
48
49
50         public void ExecuteTransaction(Transaction transaction)
51         {
52             _transactions.Add(transaction);
53             try
```

```
54         {
55             transaction.Execute();
56         }
57         catch (InvalidOperationException exception)
58         {
59             Console.WriteLine("An Error has been found in executing
60                 ↳ transaction");
61             Console.WriteLine("The error was: " + exception.Message);
62         }
63     }
64
65     public void Rollback(Transaction transaction)
66     {
67
68         transaction.Rollback();
69
70
71     }
72
73     public void PrintTransactionHistory()
74     {
75
76         for(int i = 0; i < _transactions.Count; i++)
77         {
78             Console.WriteLine("\nTransaction number is " + (i + 1));
79             _transactions[i].Print();
80
81         }
82         Console.WriteLine("Do you want to Roll back a transaction?");
83         String UserRequest = Console.ReadLine();
84         if(UserRequest == "No")
85         {
86             return;
87         }
88         if(UserRequest == "Yes")
89         {
90             try
91             {
92                 Console.WriteLine("What transaction would you like to Rollback
93                     ↳ ");
94                 String Rollbackoption = Console.ReadLine();
95                 int RollbackCall = Convert.ToInt32(Rollbackoption);
96                 BankSystem.DoRollBack(_transactions[RollbackCall - 1]);
97             }
98             catch (ArgumentOutOfRangeException exception)
99             {
100                 Console.WriteLine("An error was detected: " +
101                     ↳ exception.GetType().ToString() + "With message \"" +
102                     ↳ exception.Message + " ");
103             }
104             catch (InvalidOperationException exception)
105             {
106             }
```

```
103         Console.WriteLine("The following error was detercted: " +  
    ↪         exception.GetType().ToString() + "With message \"" +  
104             exception.Message + " ");  
105     }  
106 }  
107  
108  
109  
110 }  
111  
112  
113  
114 }  
115 }
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Connor_Gent_7._1_task
8  {
9      public class Account
10     {
11         public decimal Balance;
12         public string Name { get; set; }
13         public Account(decimal balance, string name)
14         {
15             Balance = balance;
16             Name = name;
17         }
18
19
20
21         public bool Deposit(decimal amount)
22         {
23
24             if(amount <= 0)
25             {
26                 Console.WriteLine("Deposit not successful. Please enter a valid
27                                     ↪ value");
28                 return false;
29             }
30             Balance += amount;
31             Console.WriteLine("The new balance is " + Balance);
32             return true;
33         }
34
35         public bool Withdraw(decimal amount)
36         {
37
38             if (amount <= 0 || amount > Balance)
39             {
40                 Console.WriteLine("Withdraw not successful, please enter a valid
41                                     ↪ value");
42                 return false;
43             }
44
45             Balance -= amount;
46             Console.WriteLine("The new balance is " + Balance);
47             return true;
48
49         }
50     }
51
```

```
52
53
54     public void Print()
55
56     {
57         Console.WriteLine("The balance is " + Balance);
58         Console.WriteLine("This account belongs to " + Name);
59     }
60
61     public void Quit()
62     {
63         Environment.Exit(0);
64     }
65
66
67
68     }
69 }
```