Deakin University

Object Oriented Development

OnTrack Submission

Abstract Transactions

Submitted By:
Peter Stacey
pstacey
2020/05/04 14:24

Tutor:
Dipto Pratyaksa

Outcome	**Weight**
Evaluate Code	♦♦♦◊◊
Principles	♦♦♦◊◊
Build Programs	♦♦◊◊◊
Design	♦♦◊◊◊
Justify	♦♦♦◊◊

This task involves extending the bank account by adding a base class for all transactions. That aspect requires evaluating the existing code in order to change it appropriately to integrate the new parent class into the design and derive the more specific transaction classes from the new base class. It also involves integrating the new class into the bank class and the addition of methods into the bank system. With the addition of a base class for all transactions, we have added further core concepts of OOP into the program, which aligns with the learning outcomes of the subject, and in doing so, removed some duplication across classes, by abstracting aspects the transactions into the base class. My code is evidence of meeting the required outcomes and will be supported by additional diagrams in my video.

May 4, 2020

```
using System;

namespace Task_7_1P
{
    /// <summary>
    /// Prototype for a deposit transaction
    /// </summary>
    class DepositTransaction : Transaction
    {
        // Instance variables
        private Account _account;

        public Account Account { get => _account; }

        /// <summary>
        /// Constructs a deposit transaction object
        /// </summary>
        /// <param name="account">Account to deposit into</param>
        /// <param name="amount">Amount to deposit</param>
        public DepositTransaction(Account account, decimal amount) : base(amount)
        {
            _account = account;
        }

        /// <summary>
        /// Returns the name of the account receiving the deposit
        /// </summary>
        /// <returns>
        /// String of the account name
        /// </returns>
        public override string GetAccountName()
        {
            return _account.Name;
        }

        /// <summary>
        /// Prints the details and status of a deposit
        /// </summary>
        public override void Print()
        {
            Console.WriteLine(new String('-', 85));
            Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|{3, 20}|",
                "ACCOUNT", "DEPOSIT AMOUNT", "STATUS", "CURRENT BALANCE");
            Console.WriteLine(new String('-', 85));
            Console.Write("|{0, -20}|{1, 20}|", _account.Name,
             ↪  _amount.ToString("C"));
            if (!Executed)
            {
                Console.Write("{0, 20}|", "Pending");
            }
            else if (Reversed)
            {
                Console.Write("{0, 20}|", "Deposit reversed");
```

```
53              }
54              else if (Success)
55              {
56                  Console.Write("{0, 20}|", "Deposit complete");
57              }
58              else if (!Success)
59              {
60                  Console.Write("{0, 20}|", "Invalid deposit");
61              }
62              Console.WriteLine("{0, 20}|", _account.Balance.ToString("C"));
63              Console.WriteLine(new String('-', 85));
64          }
65
66          /// <summary>
67          /// Executes a deposit transaction
68          /// </summary>
69          public override void Execute()
70          {
71              base.Execute();
72
73              _success = _account.Deposit(_amount);
74              Console.WriteLine(Success);
75              if (!_success)
76              {
77                  throw new InvalidOperationException("Deposit amount invalid");
78              }
79          }
80
81          /// <summary>
82          /// Reverses a deposit if previously executed successfully
83          /// </summary>
84          public override void Rollback()
85          {
86              base.Rollback();
87              bool complete = _account.Withdraw(_amount); // Withdraw returns boolean
88              if (!complete) // Withdraw didn't occur
89              {
90                  throw new InvalidOperationException("Insufficient funds to
                    ↪  rollback");
91              }
92              base.Reversed = true;
93          }
94      }
95  }
```

```
using System;

namespace Task_7_1P
{
    /// <summary>
    /// Prototype for a Withdraw transaction
    /// </summary>
    class WithdrawTransaction : Transaction
    {
        // Instance variables
        private Account _account;

        /// <summary>
        /// Constructs a WithdrawTransaction
        /// </summary>
        /// <param name="account">Account to withdraw from</param>
        /// <param name="amount">Amount to withdraw</param>
        public WithdrawTransaction(Account account, decimal amount) : base(amount)
        {
            _account = account;
        }

        /// <summary>
        /// Returns the name of the account being debited
        /// </summary>
        /// <returns>
        /// String of the account name
        /// </returns>
        public override string GetAccountName()
        {
            return _account.Name;
        }

        /// <summary>
        /// Prints the details and status of the withdrawal
        /// </summary>
        public override void Print()
        {
            Console.WriteLine(new String('-', 85));
            Console.WriteLine("|{0, -20}|{1, 20}|{2, 20}|{3, 20}|",
                "ACCOUNT", "WITHDRAW AMOUNT", "STATUS", "CURRENT BALANCE");
            Console.WriteLine(new String('-', 85));
            Console.Write("|{0, -20}|{1, 20}|", _account.Name,
            ↪  _amount.ToString("C"));
            if (!Executed)
            {
                Console.Write("{0, 20}|", "Pending");
            }
            else if (Reversed)
            {
                Console.Write("{0, 20}|", "Withdraw reversed");
            }
            else if (Success)
```

```
53              {
54                  Console.Write("{0, 20}|", "Withdraw complete");
55              }
56              else if (!Success)
57              {
58                  Console.Write("{0, 20}|", "Insufficient funds");
59              }
60              Console.WriteLine("{0, 20}|", _account.Balance.ToString("C"));
61              Console.WriteLine(new String('-', 85));
62          }
63
64          /// <summary>
65          /// Executes the withdrawal
66          /// </summary>
67          /// <exception cref="System.InvalidOperationException">Thrown
68          /// when the withdraw is already complete or insufficient funds</exception>
69          public override void Execute()
70          {
71              base.Execute();
72
73              _success = _account.Withdraw(_amount);
74              if (!_success)
75              {
76                  throw new InvalidOperationException("Insufficient funds");
77              }
78          }
79
80          /// <summary>
81          /// Reverses the withdraw if previously executed successfully
82          /// </summary>
83          /// <exception cref="System.InvalidOperationException">Thrown
84          /// if already rolled back or if there are insufficient
85          /// funds to complete the rollback</exception>
86          public override void Rollback()
87          {
88              base.Rollback();
89              bool complete = _account.Deposit(_amount); // Deposit returns boolean
90              if (!complete) // Deposit didn't occur
91              {
92                  throw new InvalidOperationException("Invalid amount");
93              }
94              base.Reversed = true;
95          }
96      }
97  }
```

```cs
using System;

namespace Task_7_1P
{
    /// <summary>
    /// Prototype for a transfer transaction
    /// </summary>
    class TransferTransaction : Transaction
    {
        // Instance variables
        private Account _fromAccount;
        private Account _toAccount;
        private DepositTransaction _deposit;
        private WithdrawTransaction _withdraw;

        // Properties
        public new bool Success { get => (_deposit.Success && _withdraw.Success); }

        /// <summary>
        /// Constructor for a transfer transaction
        /// </summary>
        /// <param name="fromAccount">The account to transfer from</param>
        /// <param name="toAccount">The account to transfer to</param>
        /// <param name="amount">The amount to transfer</param>
        /// <exception cref="System.ArgumentOutOfRangeException">Thrown
        /// when the amount is negative</exception>
        public TransferTransaction(Account fromAccount, Account toAccount, decimal
        ↪ amount) : base(amount)
        {
            _fromAccount = fromAccount;
            _toAccount = toAccount;
            _withdraw = new WithdrawTransaction(_fromAccount, _amount);
            _deposit = new DepositTransaction(_toAccount, _amount);
        }

        /// <summary>
        /// Returns the name of the account(s) in the transaction
        /// </summary>
        /// <returns>
        /// String of the account names
        /// </returns>
        public override string GetAccountName()
        {
            return "From: " + _fromAccount.Name + ", To: " + _toAccount.Name;
        }

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

```
103                     _deposit.Print();
104                     try
105                     {
106                         _withdraw.Rollback();
107                     }
108                     catch (InvalidOperationException e)
109                     {
110                         Console.WriteLine("Withdraw could not be reversed with
                            ↪  reason: " + e.Message);
111                         _withdraw.Print();
112                         return;
113                     }
114                 }
115             }
116             _success = true;
117         }
118 
119         /// <summary>
120         /// Rolls the transfer back
121         /// </summary>
122         /// <exception cref="System.InvalidOperationException">Thrown
123         /// when the rollback has already been executed or it fails</exception>
124         public override void Rollback()
125         {
126             base.Rollback();
127 
128             if (this.Success)
129             {
130                 try
131                 {
132                     _deposit.Rollback();
133                 }
134                 catch (InvalidOperationException exception)
135                 {
136                     Console.WriteLine("Failed to rollback deposit: "
137                         + exception.Message);
138                     return;
139                 }
140 
141                 try
142                 {
143                     _withdraw.Rollback();
144                 }
145                 catch (InvalidOperationException exception)
146                 {
147                     Console.WriteLine("Failed to rollback withdraw: "
148                         + exception.Message);
149                     return;
150                 }
151             }
152             base.Reversed = true;
153         }
154     }
```

```
155 }
```

```cs
using System;

namespace Task_7_1P
{
    /// <summary>
    /// Baseclass for transaction classes
    /// </summary>
    abstract class Transaction
    {
        // Instance variables
        protected decimal _amount;
        protected Boolean _success;
        private Boolean _executed;
        private Boolean _reversed;
        private DateTime _dateStamp;

        // public properties
        public Boolean Success { get => _success; }
        public Boolean Executed { get => _executed; }
        public Boolean Reversed { get => _reversed; set => _reversed = value; }
        public DateTime DateStamp { get => _dateStamp; }
        public decimal Amount { get => _amount; } // Added for the transaction
        ↪ history print

        /// <summary>
        /// Creates a new Transaction object
        /// </summary>
        /// <param name="amount">The amount of the transaction</param>
        public Transaction(decimal amount)
        {
            if (amount > 0)
            {
                _amount = amount;
            }
            else
            {
                amount = 0;
                throw new ArgumentOutOfRangeException("Amount must be > $0.00");
            }
            // _executed, _success, _reversed false by default
        }

        /// <summary>
        /// Provides a virtual method to return the name of the account(s)
        /// involved in the transaction
        /// </summary>
        /// <returns>
        /// String of the account name
        /// </returns>
        public virtual string GetAccountName()
        {
            return "Unknown name";
        }
```

```
53
54          /// <summary>
55          /// Writes the amount and status to the Console
56          /// </summary>
57          public virtual void Print()
58          {
59              Console.WriteLine(
60                  "Transaction amount: {0}, Executed: {1}, Success: {2}, Reversed:
                    ↪ {3}",
61                  _amount.ToString("C"), _executed, _success, _reversed);
62          }
63
64          /// <summary>
65          /// Records execution of the transaction if not previously executed
                ↪ successfully
66          /// </summary>
67          public virtual void Execute()
68          {
69              if (_executed && _success)
70              {
71                  throw new InvalidOperationException("Transaction previously
                        ↪ executed");
72              }
73              _dateStamp = DateTime.Now;
74              _executed = true;
75          }
76
77          /// <summary>
78          /// Records rolling back of the transaction if not previously rolled back
79          /// </summary>
80          public virtual void Rollback()
81          {
82              if (_reversed)
83              {
84                  throw new InvalidOperationException("Transaction already reversed");
85              }
86              else if (!_success)
87              {
88                  throw new InvalidOperationException(
89                      "Transaction not successfully executed. Nothing to rollback.");
90              }
91              _dateStamp = DateTime.Now;
92          }
93      }
94  }
```

```
using System;

namespace Task_7_1P
{
    enum MenuOption
    {
        CreateAccount,
        Withdraw,
        Deposit,
        Transfer,
        Rollback,
        Print,
        Quit
    }

    /// <summary>
    /// BankSystem implements a banking system to operate on accounts
    /// </summary>
    class BankSystem
    {
        // Reads string input in the console
        /// <summary>
        /// Reads string input in the console
        /// </summary>
        /// <returns>
        /// The string input of the user
        /// </returns>
        /// <param name="prompt">The string prompt for the user</param>
        public static String ReadString(String prompt)
        {
            Console.Write(prompt + ": ");
            return Console.ReadLine();
        }

        // Reads integer input in the console
        /// <summary>
        /// Reads integerinput in the console
        /// </summary>
        /// <returns>
        /// The input of the user as an integer
        /// </returns>
        /// <param name="prompt">The string prompt for the user</param>
        public static int ReadInteger(String prompt)
        {
            int number = 0;
            string numberInput = ReadString(prompt);
            while (!(int.TryParse(numberInput, out number)))
            {
                Console.WriteLine("Please enter a whole number");
                numberInput = ReadString(prompt);
            }
            return Convert.ToInt32(numberInput);
        }
```

```
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
                    ↪  greater");
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. New Account    *");
```

```
106             Console.WriteLine("*  2. Withdraw     *");
107             Console.WriteLine("*  3. Deposit      *");
108             Console.WriteLine("*  4. Transfer     *");
109             Console.WriteLine("*  5. Rollback     *");
110             Console.WriteLine("*  6. Print        *");
111             Console.WriteLine("*  7. Quit         *");
112             Console.WriteLine("********************");
113         }
114
115         /// <summary>
116         /// Returns a menu option chosen by the user
117         /// </summary>
118         /// <returns>
119         /// MenuOption chosen by the user
120         /// </returns>
121         private static MenuOption ReadUserOption()
122         {
123             DisplayMenu();
124             int option = ReadInteger("Choose an option", 1,
125                 Enum.GetNames(typeof(MenuOption)).Length);
126             return (MenuOption)(option - 1);
127         }
128
129         /// <summary>
130         /// Attempts to deposit funds into an account at a bank
131         /// </summary>
132         /// <param name="bank">The bank holding the account to deposit into</param>
133         static void DoDeposit(Bank bank)
134         {
135             Account account = FindAccount(bank);
136             if (account != null)
137             {
138                 decimal amount = ReadDecimal("Enter the amount");
139                 DepositTransaction deposit = new DepositTransaction(account,
                    ↪ amount);
140                 try
141                 {
142                     bank.Execute(deposit);
143                 }
144                 catch (InvalidOperationException)
145                 {
146                     deposit.Print();
147                     return;
148                 }
149                 deposit.Print();
150             }
151         }
152
153         /// <summary>
154         /// Attempts to withdraw funds from an account at a bank
155         /// </summary>
156         /// <param name="bank">The bank holding account to withdraw from</param>
157         static void DoWithdraw(Bank bank)
```

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

```
208         static void DoPrint(Bank bank)
209         {
210             Account account = FindAccount(bank);
211             if (account != null)
212             {
213                 account.Print();
214             }
215         }
216 
217         /// <summary>
218         /// Prints a list of transactions and allows them to be rolled back
219         /// if necessary
220         /// </summary>
221         /// <param name="bank">The bank to rollback transactions for</param>
222         static void DoRollback(Bank bank)
223         {
224             bank.PrintTransactionHistory();
225             int result = ReadInteger(
226                 "Enter transaction # to rollback (0 for no rollback)",
227                 0, bank.Transactions.Count);
228 
229             if (result == 0)
230                 return;
231 
232             bank.Rollback(bank.Transactions[result - 1]);
233         }
234 
235         /// <summary>
236         /// Creates a new account and adds it to the Bank
237         /// </summary>
238         /// <param name="bank">The bank to create the account in</param>
239         static void CreateAccount(Bank bank)
240         {
241             string name = ReadString("Enter account name");
242             decimal balance = ReadDecimal("Enter the opening balance");
243             bank.AddAccount(new Account(name, balance));
244         }
245 
246         private static Account FindAccount(Bank bank)
247         {
248             Account account = null;
249             string name = ReadString("Enter the account name");
250             account = bank.GetAccount(name);
251             if (account == null)
252             {
253                 Console.WriteLine("That account name does not exist at this bank");
254             }
255             return account;
256         }
257 
258         static void Main(string[] args)
259         {
260             Bank bank = new Bank();
```

```
261
262             do
263             {
264                 MenuOption chosen = ReadUserOption();
265                 switch (chosen)
266                 {
267                     case MenuOption.CreateAccount:
268                         CreateAccount(bank); break;
269
270                     case MenuOption.Withdraw:
271                         DoWithdraw(bank); break;
272
273                     case MenuOption.Deposit:
274                         DoDeposit(bank); break;
275
276                     case MenuOption.Transfer:
277                         DoTransfer(bank); break;
278
279                     case MenuOption.Rollback:
280                         DoRollback(bank); break;
281
282                     case MenuOption.Print:
283                         DoPrint(bank); break;
284
285                     case MenuOption.Quit:
286                     default:
287                         Console.WriteLine("Goodbye");
288                         System.Environment.Exit(0); // terminates the program
289                         break; // unreachable
290                 }
291             } while (true);
292         }
293     }
294 }
```

```cs
using System;
using System.Collections.Generic;

namespace Task_7_1P
{
    /// <summary>
    /// Prototype for a bank to hold accounts
    /// </summary>
    class Bank
    {
        // Instance variables
        private List<Account> _accounts;
        private List<Transaction> _transactions;

        public List<Transaction> Transactions { get => _transactions; }

        /// <summary>
        /// Creates an empty bank object with a list for accounts
        /// </summary>
        public Bank()
        {
            _accounts = new List<Account>();
            _transactions = new List<Transaction>();
        }

        /// <summary>
        /// Adds an account to the Bank accounts register
        /// </summary>
        /// <param name="account"></param>
        public void AddAccount(Account account)
        {
            _accounts.Add(account);
        }

        /// <summary>
        /// Returns the first Account corresponding to the name, or
        /// null if there is no account matching the criteria
        /// </summary>
        /// <param name="name"></param>
        /// <returns>
        /// Account matching the provided name, or null
        /// </returns>
        public Account GetAccount(string name)
        {
            foreach (Account account in _accounts)
            {
                if (account.Name == name)
                {
                    return account;
                }
            }
            return null;
        }
```

```
54
55          /// <summary>
56          /// Executes a transaction
57          /// </summary>
58          /// <param name="transaction">Transaction to execute</param>
59          public void Execute(Transaction transaction)
60          {
61              _transactions.Add(transaction);
62              try
63              {
64                  transaction.Execute();
65              }
66              catch (InvalidOperationException exception)
67              {
68                  Console.WriteLine("An error occurred in executing the transaction");
69                  Console.WriteLine("The error was: " + exception.Message);
70              }
71          }
72
73          /// <summary>
74          /// Rolls a transaction back
75          /// </summary>
76          /// <param name="transaction">Transaction to execute</param>
77          public void Rollback(Transaction transaction)
78          {
79              try
80              {
81                  transaction.Rollback();
82              }
83              catch (InvalidOperationException exception)
84              {
85                  Console.WriteLine("An error occurred in rolling the transaction
                    ↪ back");
86                  Console.WriteLine("The error was: " + exception.Message);
87              }
88          }
89
90          /// <summary>
91          /// Helper function for PrintTransactionHistory that converts the
92          /// type of the transaction to a string
93          /// </summary>
94          /// <param name="transaction">The transaction to return the
95          /// type of</param>
96          /// <returns>
97          /// The type as a string representation
98          /// </returns>
99          public string TransactionType(Transaction transaction)
100         {
101             switch (transaction.GetType().ToString())
102             {
103                 case "Task_7_1P.DepositTransaction":
104                     return "Deposit";
105                 case "Task_7_1P.WithdrawTransaction":
```

```
106                     return "Withdraw";
107                 case "Task_7_1P.TransferTransaction":
108                     return "Transfer";
109             }
110             return "Unknown";
111         }
112 
113         /// <summary>
114         /// Helper function for PrintTransactionHistory that converts the
115         /// current status to a string representation
116         /// </summary>
117         /// <param name="transaction">The transaction to return the
118         /// type of</param>
119         /// <returns>
120         /// The status as a string representation
121         /// </returns>
122         public string TransactionStatus(Transaction transaction)
123         {
124             if (!transaction.Executed)
125             {
126                 return "Pending";
127             }
128             else if (transaction.Reversed)
129             {
130                 return "Reversed";
131             }
132             else if (!transaction.Success)
133             {
134                 return "Incomplete";
135             }
136             else
137             {
138                 return "Complete";
139             }
140         }
141 
142         /// <summary>
143         /// Writes the list of transactions to the Console in a table format
144         /// </summary>
145         public void PrintTransactionHistory()
146         {
147             string transactionType = "";
148             string transactionStatus = "";
149             Console.WriteLine(new String('-', 127));
150             Console.WriteLine(
151                 "| {0,2} |{1,-25} | {2,-15}| {3,-40}|{4,15} | {5,15} |", "#",
152                 "DateTime", "Type", "Account Details", "Amount", "Status");
153             Console.WriteLine(new String('=', 127));
154             for (int i = 0; i < _transactions.Count; i++)
155             {
156                 transactionType = TransactionType(_transactions[i]);
157                 transactionStatus = TransactionStatus(_transactions[i]);
158                 Console.WriteLine(
```

```
159                     "| {0,2} |{1,-25} | {2,-15}| {3,-40}|{4,15} | {5,15} |",
160                     i + 1, _transactions[i].DateStamp, transactionType,
161                     _transactions[i].GetAccountName(),
162                     _transactions[i].Amount.ToString("C"), transactionStatus);
163             }
164             Console.WriteLine(new String('=', 127));
165         }
166     }
167 }
```

```cs
using System;

namespace Task_7_1P
{
    /// <summary>
    /// A bank account class to hold the account name and balance details
    /// </summary>
    class Account
    {
        // Instance variables
        private String _name;
        private decimal _balance;

        // Read-only properties
        public String Name { get => _name; }
        public decimal Balance { get => _balance; }


        /// <summary>
        /// Class constructor
        /// </summary>
        /// <param name="name">The name string for the account</param>
        /// <param name="balance">The decimal balance of the account</param>
        public Account(String name, decimal balance = 0)
        {
            _name = name;
            if (balance < 0)
                return;
            _balance = balance;
        }

        /// <summary>
        /// Deposits money into the account
        /// </summary>
        /// <returns>
        /// Boolean whether the deposit was successful (true) or not (false)
        /// </returns>
        /// <param name="amount">The decimal amount to add to the balance</param>
        public Boolean Deposit(decimal amount)
        {
            if ((amount < 0) || (amount == decimal.MaxValue))
                return false;

            _balance += amount;
            return true;
        }

        /// <summary>
        /// Withdraws money from the account (with no overdraw protection currently)
        /// </summary>
        /// <returns>
        /// Boolean whether the withdrawal was successful (true) or not (false)
        /// </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  }
```