Deakin University

Object Oriented Development

OnTrack Submission

Abstract Transactions

Submitted By:
Akashdeep Akashdeep
s223040483
2023/05/10 20:12

Tutor:
Son Nguyen

Outcome	**Weight**
Build Programs	♦♦♦♦◊

Learned about inheritance, polymorphism along with the use of abstract classes and method overriding

May 10, 2023

Produced by Doubtfire

```
using System;
using SplashKitSDK;

namespace BankProgram
{
    public class Program
    {
        private static Account? result;

        public enum MenuOption
        {
            Withdraw,
            Deposit,
            Transfer,
            NewAccount,
            Print,
            PrintTransactionHistory,
            Quit
        }
        public static void Main()
        {

            //Creating bank class object
            Bank bank = new Bank("Australia Bank");
            Account account = new Account("Akash Account", 50000);
            account.Print();

            // Account account2 = new Account("jake Account", 70000);

            MenuOption userSelection;

            do
            {
                userSelection = ReadUserOption();
                switch (userSelection)
                {

                    case MenuOption.Withdraw:
                        DoWithdraw(bank);
                        break;

                    case MenuOption.Deposit:
                        DoDeposit(bank);
                        break;

                    case MenuOption.Transfer:
                        DoTransfer(bank);
                        break;

                    //Adding new Menu option to create new account
                    case MenuOption.NewAccount:
                        AddAccount(bank);
                        break;
```

```
54
55                  case MenuOption.Print:
56                      DoPrint(account);
57                      break;
58
59                  case MenuOption.PrintTransactionHistory:
60                      bank.PrintTransactionHistory();
61                      break;
62
63                  case MenuOption.Quit:
64                      Console.WriteLine("Quit...");
65                      break;
66
67              }
68          }
69          while (userSelection != MenuOption.Quit);
70      }
71
72      private static MenuOption ReadUserOption()
73      {
74
75          int option = 0;
76          Console.WriteLine("Select an option [1-7]:");
77          Console.WriteLine("------------------");
78          Console.WriteLine("1: Withdraw");
79          Console.WriteLine("2: Deposit");
80          Console.WriteLine("3: Transfer");
81          //Including new menu option to display it to user
82          Console.WriteLine("4: New Account");
83          Console.WriteLine("5: Print");
84          Console.WriteLine("6: PrintTransactionHistory");
85          Console.WriteLine("7: Quit");
86          Console.WriteLine("------------------");
87
88          do
89          {
90              try
91              {
92                  option = Convert.ToInt32(Console.ReadLine());
93              }
94              catch (Exception e)
95              {
96                  Console.WriteLine(e.Message);
97                  option = -1;
98              }
99              if (option < 1 || option > 7)
100             {
101                 Console.WriteLine("Please select a valid option between 1 and
                    ↪ 7");
102             }
103         }
104
105         while (option < 1 || option > 7);
```

```
106             return (MenuOption)(option - 1);
107
108         }
109
110         //Execute the below method when user wants to deposit some amount
111         private static void DoDeposit(Bank toBank)
112         {
113             Account toAccount = FindAccount(toBank);
114             if (toAccount == null)
115             {
116                 return;
117             }
118             decimal amount = 0;
119             Console.WriteLine("How much you want to Deposit?");
120             try
121             {
122                 amount = Convert.ToDecimal(Console.ReadLine());
123                 DepositTransaction deposit = new DepositTransaction(toAccount,
                    ↪ amount);
124                 toBank.ExecuteTransaction(deposit);
125                 if (deposit.Success)
126                 {
127                     deposit.Print();
128                 }
129             }
130             catch (Exception e)
131             {
132                 Console.WriteLine(e.Message);
133
134             }
135         }
136
137         //Execute the below method when user wants to withdraw amount
138         private static void DoWithdraw(Bank toBank)
139         {
140             Account toAccount = FindAccount(toBank);
141             if (toAccount == null)
142             {
143                 return;
144             }
145
146             decimal amount = 0;
147             Console.WriteLine("How much you want to Withdraw?");
148
149             amount = Convert.ToDecimal(Console.ReadLine());
150             WithdrawTransaction withdraw = new WithdrawTransaction(toAccount,
                ↪ amount);
151             toBank.ExecuteTransaction(withdraw);
152             if (withdraw.Success)
153             {
154                 withdraw.Print();
155             }
156         }
```

```
157
158
159         ////Execute the below method when user wants to transfer amount from one
            ↪  account to another
160         private static void DoTransfer(Bank bank)
161         {
162             Console.WriteLine("Enter account to transfer from");
163             Account fromAccount = FindAccount(bank);
164             if(fromAccount == null) return;
165
166             Console.WriteLine("Enter the account to transfer to");
167             Account toAccount = FindAccount(bank);
168             if (toAccount == null)
169             {
170                 return;
171             }
172
173             decimal amount;
174             Console.WriteLine("How much amount you want to transfer?");
175
176             try
177             {
178                 amount = Convert.ToDecimal(Console.ReadLine());
179                 TransferTransaction transaction = new
                    ↪  TransferTransaction(fromAccount, toAccount, amount);
180                 bank.ExecuteTransaction(transaction);
181                 if (transaction.Success)
182                 {
183                     transaction.Print();
184                 }
185             }
186             catch (Exception e)
187             {
188                 Console.WriteLine(e.Message);
189             }
190         }
191
192         //Execute the below method when user wants to print the account details
193         private static void DoPrint(Account account)
194         {
195             account.Print();
196         }
197
198         //Execute the below method when user wants to add new account
199         private static void AddAccount(Bank bank)
200         {
201             Console.WriteLine("Please enter the account name");
202             String accountName = Convert.ToString(Console.ReadLine());
203             Console.WriteLine("Please enter the starting balance");
204
205             decimal openingBalance = Convert.ToDecimal(Console.ReadLine());
206             Account newAccount = new Account(accountName, openingBalance);
207             Console.WriteLine("newAccount name is: " + newAccount.Name + " and
                ↪  opening balance is: " + openingBalance);
```

```
208             bank.AddAccount(newAccount);
209         }
210 
211         //Execute the below method when user wants to find an account from the list
            ↪  of accounts in bank class
212         private static Account FindAccount(Bank fromBank)
213         {
214 
215             Console.Write("Enter account name:");
216             string username = Console.ReadLine();
217 
218 
219             Account newname = fromBank.GetAccount(username);
220             Account result = fromBank.GetAccount(username);
221 
222             if (result == null)
223             {
224                 Console.WriteLine($"No account found with name {username}");
225             }
226 
227             return result;
228 
229         }
230     }
231 }
```

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

public class Bank
{
    private List<Account> _accounts = new List<Account>();
    private List<Transaction> _transaction = new List<Transaction>();
    public string bankname;

    //Bank class constructor
    public Bank(string bankname)
    {
        this.bankname = bankname;
        _accounts = new List<Account>();
    }

    //Below method to add new account in list _accounts
    public void AddAccount(Account account)
    {
        if (_accounts != null)
        {
            _accounts.Add(account);
        }
    }

    //Below method to find account from the list of _accounts in bank
    public Account GetAccount(string name)
    {

        foreach (Account account in _accounts)
        {
            if (account.Name == name)
            {
                return account;
            }
        }
        return null;

    }

    public void ExecuteTransaction(Transaction transaction)
    {
        if (_transaction != null)
        {
        _transaction.Add(transaction);
        transaction.Execute();
        }
    }

    public void PrintTransactionHistory()
    {
        foreach (Transaction transaction in _transaction)
        {
```

```
54              transaction.Print();
55          }
56      }
57
58  }
```

```
using System;
using SplashKitSDK;

public class Account
{
    private decimal _balance;
    private string _name;

    //Account class constructor
    public Account(string name, decimal startingBalance)
    {
        _name = name;
        _balance = startingBalance;
    }

    //Deposit method to deposit the amount in the account and update the balance
    public bool Deposit(decimal amountToDeposit)
    {
        if (amountToDeposit > 0)
        {
            _balance += amountToDeposit;
            return true;
        }
        return false;
    }

    //Withdraw method to withdraw the amount in the account and update the balance
    public bool Withdraw(decimal amountToWidraw)
    {
        if (amountToWidraw < _balance && amountToWidraw > 0)
        {
            _balance -= amountToWidraw;
            return true;
        }
        return false;

    }
    //Read only property
    public string Name
    {
        get
        {
            return _name;
        }
    }

    //Print method to print the name and balance of the account
    public void Print()
    {
        Console.WriteLine($"The account name: {_name}");
        Console.WriteLine($"The account balance: {_balance}");
    }
}
```

```
using System;
public abstract class Transaction
{
    protected decimal _amount;
    protected bool _executed;
    protected bool _reversed;
    protected DateTime _dateStamp;


    //Transaction class constructor
    public Transaction(decimal amount)
    {
        _amount = amount;
    }

    public bool Executed
    {
        get { return _executed; }
    }

    //Example of polymorphism as we are going to override Success property in child
    ↪  classes
    public abstract bool Success
    {
        get;
    }

    public bool Reversed
    {
        get { return _reversed; }
    }

    public DateTime DataStamp
    {
        get { return _dateStamp; }
    }

    //Abstract Print method so that we can define it in child classes of
    ↪  Transaction class
    public abstract void Print();

    public virtual void Execute()
    {
        if (_executed)
        {
            throw new Exception("Cannot execute this transaction as it has already
            ↪  executed.");

        }
        _executed = true;
        _dateStamp = DateTime.Now;

    }
```

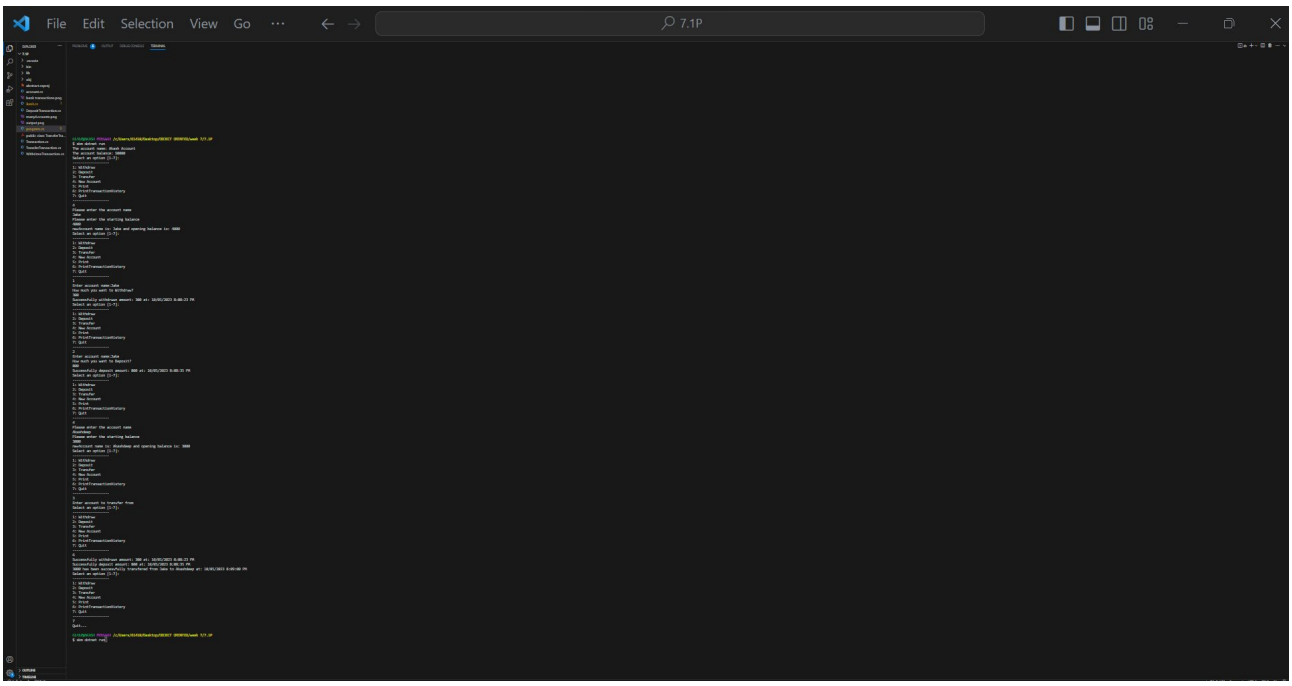
```
51
52      public virtual void Rollback()
53      {
54          if (!_executed)
55          {
56              throw new Exception("Transaction is not executed");
57          }
58
59          if (_reversed)
60          {
61              throw new Exception("Transaction has been reversed already");
62          }
63          _reversed = true;
64      }
65  }
```

```
public class WithdrawTransaction : Transaction
{
    private Account _account;
    private bool _success = false;

    //Overriding Success Property
    public override bool Success
    {
        get
        {
            return _success;
        }
    }

    public WithdrawTransaction(Account account, decimal amount) : base(amount)
    {
        _account = account;

    }

    //Overriding Execute method
    public override void Execute()
    {
        base.Execute();
        _success = _account.Withdraw(_amount);
    }

    //Overriding Rollback method
    public override void Rollback()
    {
        base.Rollback();
        _account.Deposit(_amount);
    }

    //Overriding Print method
    public override void Print()
    {
        if (_success)
        {
            //Console.WriteLine("Last transaction is successful");
             Console.WriteLine("Successfully withdrawn amount: "+_amount+" at:
             ↪  "+_dateStamp);
        }
        else if (_reversed)
        {
            Console.WriteLine("Last transaction is successfully reversed");
        }
        else
        {
            Console.WriteLine("Last transaction is not successful");
        }
    }
}
```

```
1  //DepositTransaction class is inheriting Transaction class
2  public class DepositTransaction : Transaction
3  {
4      private Account _account;
5      private bool _success = false;
6
7      //Overriding Success property showing polymorphism
8      public override bool Success
9      {
10         get
11         {
12             return _success;
13         }
14     }
15
16     public DepositTransaction(Account account, decimal amount) : base(amount)
17     {
18         _account = account;
19     }
20
21     //Overriding Execute method
22     public override void Execute()
23     {
24         base.Execute();
25         _success = _account.Deposit(_amount);
26     }
27
28     //Overriding Rollback method
29     public override void Rollback()
30     {
31         base.Rollback();
32         _account.Withdraw(_amount);
33     }
34
35     //Overriding Print method
36     public override void Print()
37     {
38         if (_success)
39         {
40             Console.WriteLine("Successfully deposit amount: "+_amount+" at:
               ↪  "+_dateStamp);
41         }
42         else if (_reversed)
43         {
44             Console.WriteLine("Last transaction is successfully reversed");
45         }
46         else
47         {
48             Console.WriteLine("Last transaction is not successful");
49         }
50     }
51
52 }
```

```
//TransferTransaction class is inheriting Transaction class
public class TransferTransaction : Transaction
{
    private Account _toAccount;
    private Account _fromAccount;
    private string _from;
    private string _to;
    private DepositTransaction _theDeposit;
    private WithdrawTransaction _theWithdraw;

    public override bool Success
    {
        get
        {
            if (_theDeposit.Success && _theWithdraw.Success)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }

    //Below method will transfer the user entered amount from "FromAccount" to
    ↪  "ToAccount" passed as parameters
    public TransferTransaction(Account FromAccount, Account ToAccount, decimal
    ↪  amount) : base(amount)
    {
        _toAccount = ToAccount;
        _fromAccount = FromAccount;
        _to = _toAccount.Name;
        _from = _fromAccount.Name;

        _theDeposit = new DepositTransaction(ToAccount, amount);
        _theWithdraw = new WithdrawTransaction(FromAccount, amount);
    }

    public override void Execute()
    {
        base.Execute();
        _theWithdraw.Execute();
        if (_theWithdraw.Success)
        {
            _theDeposit.Execute();
            if (!_theDeposit.Success)
            {
                _theWithdraw.Rollback();
                //_success = false;
            }
        }
        if (!_theWithdraw.Success)
```

```
52          {
53              Console.WriteLine($"{_fromAccount.Name} has insufficient balance");
54          }
55          _executed = true;
56
57
58      }
59
60      //Below method will rollback the last transaction
61      public override void Rollback()
62      {
63          base.Rollback();
64          _theDeposit.Rollback();
65          _theWithdraw.Rollback();
66
67          _reversed = true;
68
69      }
70
71      //Below method will print the details of last transaction
72      public override void Print()
73      {
74
75          if (_theWithdraw.Success && _theDeposit.Success)
76          {
77              Console.WriteLine(_amount + " has been successfully transfered from " +
                ↪  _from + " to " + _to+" at: "+_dateStamp);
78          }
79          else if (_reversed)
80          {
81              Console.WriteLine("Last transaction is successfully reversed");
82          }
83          else
84          {
85              Console.WriteLine("Last transaction is not successful");
86          }
87      }
88  }
```


File
Edit
Selection
View
Go
7.1P