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

Produced by Doubtfire

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connor_Gent_7._1_task
{
    class DepositTransaction: Transaction
    {
        private Account _account;




        public DepositTransaction (Account account, decimal amount) : base(amount)

        {

            this._account = account;



        }


        public override void Print()
        {
            Console.WriteLine("Transaction Successful: " + Executed + "\nDeposited
             ↪  " + _amount + " To " + _account.Name + ". Trasaction exectued at "
             ↪  + DateStamp);
        }

        public override void Rollback()
        {
            base.Rollback();

            try
            {
                if(Success == false)
                {
                    throw new InvalidOperationException("Transaction was not
                     ↪  succesful");
                }
                if (Reversed)
                {
                    throw new InvalidOperationException("Transaction again was not
                     ↪  successful");
                }
                else
                {
                    _account.Withdraw(_amount);
                    _reversed = true;
                }
            }
            catch (InvalidOperationException exception)
            {
                Console.WriteLine("There was an error detected: " +
                ↪   exception.GetType().ToString() + "With message \"" +
                ↪   exception.Message + "\"");
            }

        }

        public override void Execute()
        {
            base.Execute();
            try
            {
                if(_amount < 0 )
                {
                    throw new InvalidOperationException();

                }
                if (Executed)
                {
                    throw new InvalidOperationException();
                }
                else
                {
                    _account.Deposit(_amount);
                    _account.Deposit(_amount);
                     _success = true;
                     _excuted = true;
                    _datestamp = DateTime.Now;
                     Print();


                }
            }
            catch (InvalidOperationException)
            {
                Console.WriteLine(" Error with account. Transaction could not take
                ↪   place. ");
            }
        }
```

```



    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connor_Gent_7._1_task
{
    class WithdrawTransaction : Transaction
    {
        private Account _account;




        public WithdrawTransaction(Account account, decimal _amount) : base
         ↪  (_amount)

        {

            this._account = account;






        }


        public override void Print()
        {

            Console.WriteLine("Transaction successful: " + Executed + "\nWithdrawn:
             ↪  " + _amount
                + " from " + _account.Name + ". Transaction exectued at " +
                 ↪  DateStamp);

        }

        public override void Rollback()
        {
            base.Rollback();


                try
                {
                    if (Success == false)
                    {
                        throw new InvalidOperationException("Transaction was not
                         ↪  succesful");
                    }
                    if (_reversed)
                    {
                        throw new InvalidOperationException("Transaction again was
                        ↪  not successful");
                    }

                else
                {
                    _account.Balance += _amount;
                    _reversed = true;
                }
            }

            catch (InvalidOperationException)
            {
                Console.WriteLine("Transaction could not go forward. Check account
                ↪  to fix " + GetType().ToString());
            }

        }

        public override void Execute()
         {
            base.Execute();

            try

            {

                if (_amount > _account.Balance)
                {

                    throw new InvalidOperationException("Insufficient funds");
                }

                if (Executed)
                {

                    throw new InvalidOperationException("Transaction already
                    ↪  attempted");

                }
                if (_amount < 0)
                {
                    throw new InvalidOperationException("Please enter a valid
                    ↪  amount");

                }
                else
                {
                    _account.Withdraw(_amount);
                    _success = true;
                    _excuted = true;
                    _datestamp = DateTime.Now;
                    Print();
                }
            }
            catch (InvalidOperationException)
            {
                Console.WriteLine("Transaction could not go forward. Check account
                ↪  to fix " + GetType().ToString());
            }

        }

    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connor_Gent_7._1_task
{
    class TransferTransaction : Transaction
    {
        private Account _toaccount, _fromaccount;

        private DepositTransaction _deposit;

        private WithdrawTransaction _withdraw;





        public TransferTransaction(Account toaccount, Account fromaccount, decimal _amount) : base(_amount) //constructor

        {

            this._toaccount = toaccount;

            this._fromaccount = fromaccount;

            this._amount = _amount;



        }

        public override void Print()
        {


            Console.WriteLine("Transaction Successful: " + Executed + "\nTransferred: " + _amount
                + " from " + _fromaccount.Name + " To " + _toaccount.Name + DateStamp);
        }

        public override void Rollback()
        {
            base.Rollback();


            try

            {
                if (Success == false)
                {
                    throw new InvalidOperationException("Transaction was not succesful");
                }
                if (Reversed)
                {
                    throw new InvalidOperationException("Transaction again was not successful");
                }
                if (_amount > _toaccount.Balance)
                {
                    throw new InvalidOperationException("The " + _toaccount.Name + "Account does not have enough money");
                }
                else
                {
                    _deposit.Rollback();
                    _withdraw.Rollback();
                    _reversed = true;
                }

            }

            catch (InvalidOperationException)

            {

                Console.WriteLine("Transaction could not be completed with amount of funds in the account " + GetType().ToString());

            }

        }

        public override void Execute()

        {
            base.Execute();

            try
            {
                if (_amount > _fromaccount.Balance)
                {
                    throw new InvalidOperationException("Insufficient funds");


                }
                if (Executed)
                {
                    throw new InvalidOperationException("Transaction already attempted");
                }
                else
                {
                    _withdraw = new WithdrawTransaction(_fromaccount, _amount);
                    _withdraw.Execute();

                    if (_withdraw.Success)
                    {
                        _deposit = new DepositTransaction(_toaccount, _amount);
                        _deposit.Execute();

                        if (_deposit.Success)
                        {
                            _success = true;
                            _excuted = true;
                            _datestamp = DateTime.Now;


                            Print();
                        }
                        else
                        {
                            Rollback();
                        }
                    }
                }
            }
            catch (InvalidOperationException)
            {
                Console.WriteLine("Transaction could not be completed with amount
                 ↪  of funds in the account");
            }
        }
    }
}
```

```
using System;

namespace Connor_Gent_7._1_task
{
    abstract class Transaction
    {
        protected decimal _amount;
        protected bool _success;
        public bool _excuted, _reversed;
        public DateTime _datestamp;

        public bool Success { get => _success; }

        public bool Executed { get => _excuted; }

        public bool Reversed { get => _reversed; }

        public DateTime DateStamp { get => _datestamp; }

        public decimal Amount { get => _amount; }

        public Transaction(decimal amount)
        {
            this._amount = amount;
        }

        abstract public void Print();

        public virtual void Execute()
        {

        }

        public virtual void Rollback()
        {
            if (_reversed)
            {
                throw new InvalidOperationException("Transaction already reversed");

            }
            else if (!_success)
            {
                throw new InvalidOperationException("Transaction not successful.
                 ↪  Nothing to Rollback");

            }

            _datestamp = DateTime.Now;
        }



    }
}
```

```
using System;

namespace Connor_Gent_7._1_task
{
    public enum MenuOption

    {

        Withdraw = 1,

        Deposit = 2,

        Transfer = 3,

        AddAccount = 4,

        FindAccount = 5,

        PrintTransactionHistory = 6,

        Print = 7,

        Quit = 8

    }
    class BankSystem
    {
        static MenuOption ReadUserOption()

        {

            int choice = 0;

            do

            {

                Console.WriteLine("1. Withdraw");

                Console.WriteLine("2. Deposit");

                Console.WriteLine("3. Transfer");

                Console.WriteLine("4. Add new account");

                Console.WriteLine("5. Find Account");

                Console.WriteLine("6. Print transaction account");

                Console.WriteLine("7. Print");

                Console.WriteLine("8. Quit");

                Console.WriteLine("Enter choice: ");

                try

                {

                    choice = Convert.ToInt32(Console.ReadLine());

                }

                catch (Exception) { }

            } while (choice < 1 || choice > 8);

            return (MenuOption)choice;

        }

        public static Bank bank = new Bank();

        static void Main(string[] args)

        {



            Account Jason = new Account(420, "Jason");

            bank.AddAccount(Jason);

            Jason.Deposit(200);

            Jason.Withdraw(500);

            Jason.Print();


            Account James = new Account(420, "James");

            bank.AddAccount(James);

            James.Deposit(300);

            James.Withdraw(40);

            James.Print();

            while (true)

            {

                switch (ReadUserOption())

                {

                    case MenuOption.Withdraw:

                        DoWithdraw(bank);

                        break;

                    case MenuOption.Deposit:

                        DoDeposit(bank);

                        break;

                    case MenuOption.Transfer:

                        DoTransfer(bank);

                        break;

                    case MenuOption.AddAccount:

                       bank.AddAccount(GetAccount());

                        break;
                    case MenuOption.FindAccount:
                        FindAccount(bank);
                        break;
                    case MenuOption.PrintTransactionHistory:
                        DoPrintTransactionHistory(bank);
                        break;

                    case MenuOption.Print:

                        DoPrint(bank);

                        break;

                    case MenuOption.Quit:

                        Environment.Exit(0);

                        break;

                    default:
                        Jason.Quit();
                        break;



                }

            }
```

```

        }

        static Account GetAccount()

        {

            Console.WriteLine("Enter account name: ");

            String name = Console.ReadLine();

            Console.WriteLine("Enter starting balance: ");

            decimal balance = Convert.ToDecimal(Console.ReadLine());

            return new Account(balance, name);

        }

        static Account FindAccount(Bank bank)

        {

            Console.WriteLine("Enter account name: ");

            string name = Console.ReadLine();

            var account = bank.GetAccount(name);

            if (account == null)

            {

                Console.WriteLine("Account wiht name " + name + " not found");

            }

            return account;

        }

        static void DoWithdraw(Bank bank)

        {

            var account = FindAccount(bank);

            if (account == null)

                return;

            Console.WriteLine("Enter value: ");

            decimal amount = Convert.ToDecimal(Console.ReadLine());

            WithdrawTransaction withdrawTransaction = new
            ↪  WithdrawTransaction(account, amount);

            bank.ExecuteTransaction(withdrawTransaction);

        }

        static void DoDeposit(Bank bank)

        {

            var account = FindAccount(bank);

            if (account == null)

                return;

            Console.WriteLine("Enter value: ");

            decimal amount = Convert.ToDecimal(Console.ReadLine());

            DepositTransaction depositTransaction = new DepositTransaction(account,
            ↪  amount);

            bank.ExecuteTransaction(depositTransaction);

        }

        static void DoPrintTransactionHistory(Bank bank)
        {
            bank.PrintTransactionHistory();
        }

        static void DoTransfer(Bank bank)

        {

            Console.WriteLine("From Account:");

            var fromAccount = FindAccount(bank);

            if (fromAccount == null)

                return;

            Console.WriteLine("To Account:");

            var toAccount = FindAccount(bank);

            if (toAccount == null)

            {

                return;

            }

            Console.WriteLine("Enter amount: ");

            decimal amount = Convert.ToDecimal(Console.ReadLine());

            TransferTransaction transferTransaction = new
             ↪  TransferTransaction(toAccount, fromAccount, amount);

            bank.ExecuteTransaction(transferTransaction);

        }

        static void DoPrint(Bank bank)

        {

            var account = FindAccount(bank);

            if (account != null)

            {

                account.Print();

            }

            else

            {

                Console.WriteLine("Account not found");

            }



        }

        public static void DoRollBack(Transaction transaction)
        {
            transaction.Rollback();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connor_Gent_7._1_task
{
    class Bank
    {
        private List<Account> accountList;


        private List<Transaction> _transactions;

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



        public Bank()

        {

            accountList = new List<Account>();
            _transactions = new List<Transaction>();

        }



        public void AddAccount(Account account)

        {

            accountList.Add(account);

        }



        public Account GetAccount(String name)

        {
            return accountList.FirstOrDefault(a => a.Name == name);



        }

        public void ExecuteTransaction(Transaction transaction)
        {
            _transactions.Add(transaction);
            try
            {
                transaction.Execute();
            }
            catch (InvalidOperationException exception)
            {
                Console.WriteLine("An Error has been found in executing
                ↪  transaction");
                Console.WriteLine("The error was: " + exception.Message);
            }

        }

        public void Rollback(Transaction transaction)
        {

                transaction.Rollback();


        }

        public void PrintTransactionHistory()
        {

            for(int i = 0; i < _transactions.Count; i++)
            {
                Console.WriteLine("\nTransaction number is " + (i + 1));
                _transactions[i].Print();

            }
            Console.WriteLine("Do you want to Roll back a transaction?");
            String UserRequest = Console.ReadLine();
            if(UserRequest == "No")
            {
                return;
            }
            if(UserRequest == "Yes")
            {
                try
                {
                    Console.WriteLine("What transaction would you like to Rollback
                    ↪  ");
                    String Rollbackoption = Console.ReadLine();
                    int RollbackCall = Convert.ToInt32(Rollbackoption);
                    BankSystem.DoRollBack(_transactions[RollbackCall - 1]);
                }
                catch(ArgumentOutOfRangeException exception)
                {
                    Console.WriteLine("An error was detected: " +
                    ↪  exception.GetType().ToString() + "With message \"" +
                    ↪  exception.Message + " ");
                }
                catch(InvalidOperationException exception)
                {
                    Console.WriteLine("The following error was detercted: " +
                     ↪  exception.GetType().ToString() + "With message \"" +
                        exception.Message + " ");
                }
            }



        }



    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Connor_Gent_7._1_task
{
    public class Account
    {
        public decimal Balance;
        public string Name { get; set; }
        public Account(decimal balance, string name)
        {
            Balance = balance;
            Name = name;
        }



        public bool Deposit(decimal amount)
        {

            if(amount <= 0)
            {
                Console.WriteLine("Deposit not successful. Please enter a valid
                 ↪  value");
                return false;
            }
            Balance += amount;
            Console.WriteLine("The new balance is " + Balance);
            return true;

        }

        public bool Withdraw(decimal amount)

        {
            if (amount <= 0 || amount > Balance)
            {
                Console.WriteLine("Withdraw not successful, please enter a valid
                 ↪  value");

                return false;
            }

            Balance -= amount;
            Console.WriteLine("The new balance is " + Balance);
            return true;


        }



        public void Print()

        {
            Console.WriteLine("The balance is " + Balance);
            Console.WriteLine("This account belongs to " + Name);
        }

        public void Quit()
        {
            Environment.Exit(0);
        }



    }
}
```