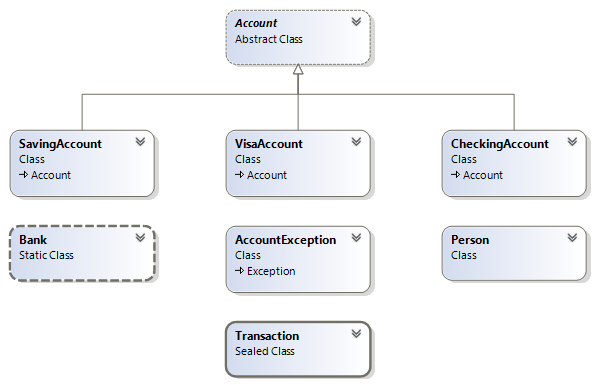
Programming II

# Implementing a Banking Application

### Due: Demonstration is due at the beginning of the second class in Week 7



The application is by far the most complex system that you have attempted so far. It consists of seven classes linked in varying degrees of tightness. The Bank class is the main driver of the application. It stores a collection of Accounts and Person. You will implement the entire system in Visual Studio. A short description of the classes with their members is given below.

You are advised to implement each type in the order that they are described.

Each type must be in separate files.

# Person class

You will implement the Person Class in Visual Studio. A short description of each class member is given below:

|  |
| --- |
| **Person**  **class** |
| **Fields** |
| **- «property setter absent» password : string**  **+ «property setter absent» SIN : double** |
| **Properties** |
| **+ «property private setter» IsAuthenticated : bool**  **+ «property setter absent» Name : string** |
| **Methods** |
| **+ «constructor» Person( name : string, sin : string )**  **+ Login(password : string) : void**  **+ Logout() : void**  **+ ToString() : string** |

### Fields:

1. **Password** – this string field represents the password of this person. (N.B. Password are not normally stored as text but as a hash value. A hashing function operates on the password and the result is stored in the field. When a user supplies a password it is passed through the same hashing function and the result is compared to the value of the field.). This is a private member.
2. **SIN** – this string field represents the sin number of the person. This member is readonly and public.

### Properties:

1. **IsAuthenticated** – this property is a bool representing if this person is logged in with the correct password. This is modified in the **Login()** and the **Logout()** methods. This is an auto-implemented property, and the getter is public and setter is private
2. **Name** – this property is a string representing the name of the person. This is an auto-implemented property, and the getter is public and setter is private.

### Methods:

1. **public Person(string name, string sin)** – This public constructor takes two parameters: a string representing the name of the person and another string representing the SIN of this person. It does the following:
   1. The method assigns the arguments to the appropriate fields.
   2. It also sets the password to the first three letters of the SIN. [use SIN.Substring(0, 3)]
2. **public void Login(string password)** – This method takes a string parameter representing the password and does the following:  
   Throw an **AccountException** object this the if the argument does not match the password  
   If the argument matches the password the IsAuthenticated property is set to true   
   This method does not display anything
3. **public void Logout()** – This is public method does not take any parameters nor does it return a value.  
   This method sets the IsAuthenticated property to false  
   This method does not display anything
4. **public override string ToString()**– This public method overrides the same method of the Object class. It does not take any parameter but returns a string representing the name of the person and if he is authenticated or not.  
   Do not display SIN and especially password.

# Transaction struct

You will implement the Transaction Class in Visual Studio. The only purpose of this class is to capture the data values for each transaction. A short description of the class members is given below:

All the properties are public with the setter absent

|  |
| --- |
| **Transaction**  **struct** |
| **Fields** |
|  |
| **Properties** |
| **+ «property setter absent» AccountNumber : string**  **+ «property setter absent» Amount : double**  **+ «property setter absent» EndBalance : double**  **+ «property setter absent» Originator : Person**  **+ «property setter absent» Time : DateTime** |
| **Methods** |
| **+ «constructor» Transaction( … )**  **+ ToString() : string** |

### Fields:

There are no fields in this struct

### Properties:

1. **AccountNumber** – this property is a string representing the account number associated with this transaction. This property is public with the setter absent.
2. **Amount** – this property is a double representing the account of this transaction. This property is public with the setter absent.
3. **EndBalance** – this property is a double representing the resulting balance at the end of this transaction. This property is public with the setter absent.
4. **Originator** – this property is a person representing the person initiating this transaction. This property is public with the setter absent.
5. **Time** – this property is a DateTime representing the time associated with this transaction. This property is public with the setter absent.

### Methods:

1. **public Transaction(string accountNumber, double amount, double endBalance, Person person, DateTime time)** – This public constructor takes five arguments. It assigns the arguments to the appropriate fields.
2. **public override string ToString()** – This method overrides the same method of the Object class. It does not take any parameter and it returns a string representing the account number, name of the person, the amount and the time that this transition was completed. [you may use **ToShortTimeString()** method of the **DateTime** class]. You **must** include the word Deposit or Withdraw in the output.

In this situation a stuct is better type than a class.

# AccountException class

You will implement the AccountException Class in Visual Studio. This inherits from the Exception Class to provide a custom exception object for this application. It consists of seven strings and two constructors:

|  |
| --- |
| **AccountException**  **Class 🠊 Exception** |
| **Fields** |
| **+ «constant» ACCOUNT\_DOES\_NOT\_EXIST : string**  **+ «constant» CREDIT\_LIMIT\_HAS\_BEEN\_EXCEEDED : string**  **+ «constant» NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT : string**  **+ «constant» NO\_OVERDRAFT : string**  **+ «constant» PASSWORD\_INCORRECT : string**  **+ «constant» USER\_DOES\_NOT\_EXIST : string**  **+ «constant» USER\_NOT\_LOGGED\_IN : string** |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» AccountException( )**  **+ «constructor» AccountException( reason : string)** |

## Fields:

All seven fields are public constants. You decide on the contents of these string constants. It MUST convey some meaningful information to the user.

1. **ACCOUNT\_DOES\_NOT\_EXIST** – this public constant string contains the condition associated with this condition.
2. **CREDIT\_LIMIT\_HAS\_BEEN\_EXCEEDED** – this public constant string contains the condition associated with this condition.
3. **NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT** – this public constant string contains the condition associated with this condition.
4. **NO\_OVERDRAFT** – this public constant string contains the condition associated with this condition.
5. **PASSWORD\_INCORRECT** – this public constant string contains the condition associated with this condition.
6. **USER\_DOES\_NOT\_EXIST** – this public constant string contains the condition associated with this condition.
7. **USER\_NOT\_LOGGED\_IN** – this public constant string contains the condition associated with this condition.

## Methods:

1. **public AccountException()**– this public constructor simply invokes the base constructor
2. **public AccountException(string reason)**– this public constructor simply invokes the base constructor with the provided argument.

# Account class

You will implement the Account Class in Visual Studio. This is an abstract class that will serve as the base class for the Visa, Checking and Saving classes. Because this is an abstract class, you may not instantiate this class. A short description of the class members is given below:

|  |
| --- |
| **Account**  **Abstract Class** |
| **Fields** |
| **-$ CURRENT\_NUMBER : int**  **# transactions : List<Transaction>**  **# holders : List<Person>**  **+ «readonly» Number : string** |
| **Properties** |
| **+ «property protected setter» Balance : double**  **+ «property protected setter» LowestBalance : double** |
| **Methods** |
| **+ «constructor» Account(type : string, balance : double)**  **+ AddUser(person : Person) : void**  **+ Deposit(amount : double, person : Person) : void**  **+ IsHolder(name : string) : bool**  **+ PrepareMonthlyReport() : void**  **+ ToString() : string** |

## Fields:

1. **holders** – this field is a list of persons representing the user who have access to this account. This is readonly and public public and is initialized at declaration.
2. **transactions** – this field is a list of transaction representing the deposits, withdrawal, payments and purchases of this account. This is readonly and public and is initialized at declaration.
3. **Number** – this property is a string representing the account number of this account. This is readonly and public.
4. **LAST\_NUMBER** – this private field is a class variable of type int. It represents the last account number that was used to generate the unique account number

## Properties:

1. **Balance** – this property is a double representing the amount in this account. This is modified in the **Deposit()** method and in the **PrepareMonthlyReport()** method of the derived classes. This is an auto-implemented property the getter is public and the setter is protected
2. **LowestBalance** – this property is a double representing the lowest balance that this amount ever drops to. This is modified in the **Deposit()** method. This is an auto-implemented property the getter is public and the setter is protected

### Methods:

1. **public Account(string type, double balance)** – This is the public constructor. It takes two parameters: a two-letter string representing the type of the account **(“VS”**, **“SV”** and **“CK”** for visa, saving and checking account respectively) and a double representing the starting balance. The method do the following:
   1. Assigns the second argument to Balance and LowestBalance
   2. Sets the Number property to the concatenation (joining) of the first argument and the class variable LastNumber
   3. Increments the class variable LastNumber

This constructor is called in the constructors on the three derived classes

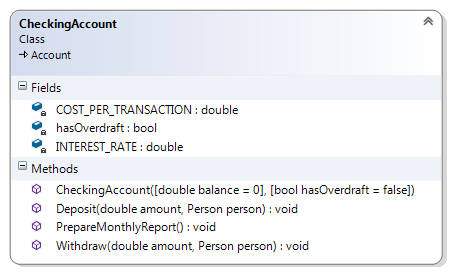
1. **public void AddUser(Person person)** – This public method takes a person object as a parameter. It adds the argument to holders (the list of persons). This method does not return a value nor does it display anything on the screen
2. **public void Deposit(double amount, Person person)** – This public method take a double parameter representing the amount to change balance by and a person object representing the person who is performing this transaction. This method does the following:
   1. Increase (or decrease) Balance by the amount specified by its argument
   2. Update LowestBalance based on the current value of Balance
   3. Create a Transaction object based on the current time (use DateTime.Now), the AccountNumber, the amount (specified by the argument), a person object (as specified by the argument and the Balance
   4. Adds the above object to the list of **transactions**

This is method is called by the **Deposit** and **Withdraw** method s of the **CheckingAccount** and **SavingsAccount** class as well as the **DoPurchase** and **DoPayment** of the **VisaAccount** class.

This method does not display anything nor does it return a value

1. **public bool IsHolder(string name)** – This public method that takes a string parameter representing the name of the user and returns true if the argument matches the name of a person in holders (the list of persons) and false otherwise.  
   You cannot use the Contains method of the list class because the list is a list of persons and not a list of strings. You will have to use a loop to check each person in the collection.  
   It does not display anything on screen
2. ***public abstract void PrepareMonthlyReport()****– This abstract public method does not take any parameter nor does it return a value.  
   Research how to declare an abstract method.  
   This method is implemented in the classes derived from Accounts.*
3. **public override string ToString()** – This method overrides the same method of the Object class. It does not take any parameter but returns a string representation of this account. It does the following:
   1. Declare and initialise a string variable to store the return value and add the following to it:
   2. Add the AccountNumber to the result value
   3. Adds the names of each of the users of the account
   4. Adds the Balance
   5. Adds all the transactions

# CheckingAccount class

You will implement the CheckingAccount Class in Visual Studio. This is a sub class is derived from the Account class. A short description of the class members is given below:

## Fields:

1. **COST\_PER\_TRANSACTION** – this is a class variable of type double representing the unit cost per transaction. All of the objects on this class will have the same value. This class variable is initialized to **0.05**.
2. **INTEREST\_RATE** – this is a class variable of type double representing the annual interest rate. All of the objects on this class will have the same value. This class variable is initialized to **0.005**.
3. **hasOverdraft** – this is a bool indicating if the balance on this account can be less than zero. This private instance variable is set in the constructor.

## Methods:

1. **public CheckingAccount(double balance = 0, bool hasOverdraft = false)** – This public constructor takes a parameter of type double representing the starting balance of the account and a bool indicating if this account has over draft permission. The constructor does the following:
   1. It invokes the base constructor with the string “CK-” and the argument.
   2. Assigns the hasOverdraft argument to the appropriate field.
2. **public new void Deposit(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be deposited and a person object representing the person do the transaction. The method does the following:
   1. If the person is registered to this account then it calls the **Deposit()** method of the base class with the appropriate arguments
3. **public void Withdraw(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be withdrawn and a person object representing the person do the transaction. The method does the following:
   1. Throws an **AccountException** object if this person in not associated with this account.
   2. Throws an **AccountException** object if this person in not logged in.
   3. Throws an **AccountException** object if the withdrawal amount is greater than the balance and there is no overdraft facility.
   4. Otherwise it calls the **Deposit()** method of the base class with the appropriate arguments (you will send negative of the amount)
4. **public override void PrepareMonthlyReport()** – this public method override the method of the base class with the same name. The method does the following:
   1. Calculate the service charge by multiplying the number of transactions by the **COST\_PER\_TRANSACTION** (how can you find out the number of transactions?)
   2. Calculate the interest by multiplying the **Balance** by the **INTEREST\_RATE** and then dividing by 12
   3. Update the **Balance** by adding the interest and subtracting the service charge
   4. **transactions** is re-initialized (use the **Clear()** method of the list class)

This method does not take any parameter nor does it display anything

# SavingAccount class

You will implement the SavingAccount Class in Visual Studio. This is a sub class derived from the Account class. A short description of the class members is given below:

|  |
| --- |
| **SavingAccount**  **Class 🠊 Account** |
| **Fields** |
| **-$ COST\_PER\_TRANSACTION : double**  **-$ INTEREST\_RATE : double** |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» SavingAccount(balance : double)**  **+ Deposit(amount: double, person Person): : void**  **+ Withdraw(amount: double, person Person): : void**  **+ PrepareMonthlyReport(): : void** |

## Fields:

1. **COST\_PER\_TRANSACTION** – this is a class variable of type double representing the unit cost per transaction. All of the objects on this class will have the same value. This class variable is initialized to **0.05**.
2. **INTEREST\_RATE** – this is a class variable of type double representing the annual interest rate. All of the objects on this class will have the same value. This class variable is initialized to **0.015**.

## Methods:

1. **public SavingAccount(double balance = 0, bool hasOverDraft = false)** – This public constructor takes a parameter of type double representing the starting balance of the account and a bool indicating if there is overdraft on this account. The constructor does the following:
   1. It invokes the base constructor with the string “SV-” and the argument.
2. **public new void Deposit(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be deposited and a person object representing the person do the transaction. The method does the following:
   1. If the person is registered to this account then it calls the **Deposit()** method of the base class with the appropriate arguments
3. **public void Withdraw(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be withdrawn and a person object representing the person do the transaction. The method does the following:
   1. Throws an **AccountException** object if this person in not associated with this account.
   2. Throws an **AccountException** object if this person in not logged in.
   3. Throws an **AccountException** object if the intended withdrawal amount exceeds the balance.
   4. Otherwise it calls the **Deposit()** method of the base class with the appropriate arguments (you will send negative of the amount)
4. **public override void PrepareMonthlyReport()** – this public method override the method of the base class with the same name. The method does the following:
   1. Calculate the service charge by multiplying the number of transactions by the **COST\_PER\_TRANSACTION** (how can you find out the number of transactions?)
   2. Calculate the interest by multiplying the **LowestBalance** by the **INTEREST\_RATE** and then dividing by 12
   3. Update the **Balance** by adding the interest and subtracting the service charge
   4. **transactions** is re-initialized (use the **Clear()** method of the list class)

This method does not take any parameter nor does it display anything

# VisaAccount class

You will implement the VisaAccount Class in Visual Studio. This is a sub class derived from the abstract Account class. A short description of the class members is given below:

|  |
| --- |
| **VisaAccount**  **Class 🠊 Account** |
| **Fields** |
| **-$ INTEREST\_RATE : double**  **- credilLimit : double** |
| **Properties** |
|  |
| **Methods** |
| **+ «constructor» VisaAccount(balance : double, creditLimit : double)**  **+ DoPayment(amount: double, person Person): : void**  **+ DoPurchase(amount: double, person Person): : void**  **+ PrepareMonthlyReport(): : void** |

## Fields:

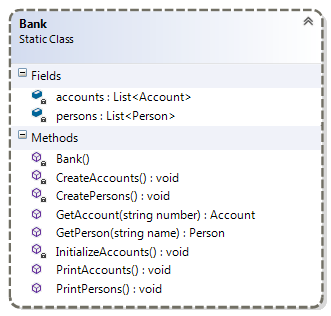
1. **creditLimit** – this is a double representing the maximum balance allowable on this account. This private instance variable is set in the constructor.
2. **INTEREST\_RATE** – this is a class variable of type double representing the annual interest rate. All of the objects on this class will have the same value. This class variable is initialized to **0.1995**.

## Methods:

1. **public VisaAccount(double balance = 0, double creditLimit = 1200)** – This public constructor takes a parameter of type double representing the starting balance of the account and a double representing the credit limit of this account. The constructor does the following:
   1. It invokes the base constructor with the string “VS-” and the argument
   2. Assigns the argument creditLimit to the appropriate fields
2. **public void DoPayment(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be deposited and a person object representing the person do the transaction. The method calls the **Deposit()** method of the base class with the appropriate arguments
3. **public void DoPurchase(double amount, Person person)** – this public method takes two arguments: a double representing the amount to be withdrawn and a person object representing the person do the transaction. The method does the following:
   1. Throws an **AccountException** object if this person in not associated with this account.
   2. Throws an **AccountException** object if this person in not logged in.
   3. Throws an **AccountException** object if this intended purchase will exceed the credit limit.
   4. Otherwise it calls the **Deposit()** method of the base class with the appropriate arguments (you will send negative of the amount)
4. **public override void PrepareMonthlyReport()** – this public method override the method of the base class with the same name. The method does the following:
   1. Calculate the interest by multiplying the LowestBalance by the InterestRate and then dividing by 12
   2. Update the Balance by subtraction the interest
   3. Transactions is re-initialized

This method does not take any parameter nor does it display anything

# Bank class

You will implement the Bank Class in Visual Studio. This is a static class where all its members are also static. [All the members of a static class must also be declared static.] A short description of the class members is given below:

## Properties:

1. **accounts** – this class variable is a list of account. This private member is initialized in the **CreateAccounts()** method.
2. **persons** – this class variable is a list of person. This private member is initialized in the **CreatePersons()** method.

## Methods:

1. **static Bank()** – This static constructor calls the private class methods **InitializeAccounts()**.

Static constructors do not take any parameters and cannot be decorated with the public or protected or private keywords

1. **static void CreatePersons()** – this private static method is given below:

persons = new List<Person>(){

new Person("Narendra", "1234-5678"),

new Person("Ilia", "2345-6789"),

new Person("Tom", "3456-7890"),

new Person("Syed", "4567-8901"),

new Person("Arben", "5678-9012"),

new Person("Patrick", "6789-0123"),

new Person("Yin", "7890-1234"),

new Person("Hao", "8901-2345"),

new Person("Jake", "9012-3456")

};

1. **static void CreateAccounts()** – this private static method is given below:

accounts = new List<Account>{

new VisaAccount(), //VS-100000

new VisaAccount(550, -500), //VS-100001

new SavingAccount(5000), //SV-100002

new SavingAccount(), //SV-100003

new CheckingAccount(2000), //CK-100004

new CheckingAccount(1500, true) //CK-100005

};

1. **static void Initialize()** – this public static method is called in the static constructor and is given below

static void InitializeAccounts()

{

CreateAccounts();

CreatePersons();

accounts[0].AddUser(persons[0]);

accounts[0].AddUser(persons[1]);

accounts[0].AddUser(persons[2]);

accounts[1].AddUser(persons[3]);

accounts[1].AddUser(persons[4]);

accounts[1].AddUser(persons[2]);

accounts[2].AddUser(persons[0]);

accounts[2].AddUser(persons[5]);

accounts[2].AddUser(persons[6]);

accounts[3].AddUser(persons[5]);

accounts[3].AddUser(persons[6]);

accounts[4].AddUser(persons[5]);

accounts[4].AddUser(persons[7]);

accounts[4].AddUser(persons[8]);

accounts[5].AddUser(persons[5]);

accounts[5].AddUser(persons[6]);

}

1. **public static void PrintAccounts()** – this public static method displays all the accounts in the accounts collection
2. **public static void PrintPersons()** – this public static method displays all the persons in the persons collection
3. **public static Person GetPerson(string name)** – this public static method takes a string representing the name of a person and returns the matching person object. The method does the following:
   1. Using a suitable loop iterate thru the static collection persons.
   2. If the person name matches the argument then this person object is returned
   3. If none of name matches, then an **AccountException** object is thrown

This method does not display anything on screen

1. **public static Account GetAccount(string number)** – this public static method takes a string representing an account number and returns the matching account. The method does the following:
   1. Using a suitable loop iterate thru the list of account
   2. If the account number matches the argument then return this account
   3. If none of the number matched, then an **AccountException** object is thrown

## Testing

Use the following code in your test harness.

Suggested code for Setting up the Bank:

//testing the visa account

Console.WriteLine("\nAll acounts:");

Bank.PrintAccounts();

Console.WriteLine("\nAll Users:");

Bank.PrintPersons();

Person p0, p1, p2, p3, p4, p5, p6, p7, p8;

p0 = Bank.GetPerson("Narendra");

p1 = Bank.GetPerson("Ilia");

p2 = Bank.GetPerson("Tom");

p3 = Bank.GetPerson("Syed");

p4 = Bank.GetPerson("Arben");

p5 = Bank.GetPerson("Patrick");

p6 = Bank.GetPerson("Yin");

p7 = Bank.GetPerson("Hao");

p8 = Bank.GetPerson("Jake");

p0.Login("123"); p1.Login("234");

p2.Login("345"); p3.Login("456");

p4.Login("567"); p5.Login("678");

p6.Login("789"); p7.Login("890");

//a visa account

VisaAccount a = (VisaAccount)Bank.GetAccount("VS-100000");

a.DoPayment(1500, p0);

a.DoPurchase(200, p1);

a.DoPurchase(25, p2);

a.DoPurchase(15, p0);

a.DoPurchase(39, p1);

a.DoPayment(400, p0);

Console.WriteLine(a);

a = (VisaAccount)Bank.GetAccount("VS-100001");

a.DoPurchase(25, p3);

a.DoPurchase(20, p4);

a.DoPurchase(15, p2);

a.DoPayment(400, p0);

Console.WriteLine(a);

//a saving account

SavingAccount b = (SavingAccount)Bank.GetAccount("SV-100002");

b.Withdraw(300, p0);

b.Withdraw(32.90, p6);

b.Withdraw(50, p5);

b.Withdraw(111.11, p5);

Console.WriteLine(b);

b = (SavingAccount)Bank.GetAccount("SV-100003");

b.Deposit(300, p3); //ok even though p3 is not a holder

b.Deposit(32.90, p2);

b.Deposit(50, p5);

b.Withdraw(111.11, p5);

Console.WriteLine(b);

//a checking account

CheckingAccount c = (CheckingAccount)Bank.GetAccount("CK-100005");

c.Deposit(33.33, p7);

c.Deposit(40.44, p7);

c.Withdraw(450, p5);

c.Withdraw(500, p5);

c.Withdraw(645, p5);

c.Withdraw(850, p6);

Console.WriteLine(c);

c = (CheckingAccount)Bank.GetAccount("CK-100004");

c.Deposit(33.33, p7);

c.Deposit(40.44, p7);

c.Withdraw(150, p5);

c.Withdraw(200, p7);

c.Withdraw(645, p7);

c.Withdraw(350, p5);

Console.WriteLine(c);

Console.WriteLine("\n\nExceptions:");

//The following will cause exception

try

{

p8.Login("911");//incorrect password

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

p3.Logout();

a.DoPurchase(12.5, p3); //exception user is not logged in

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

a.DoPurchase(12.5, p0); //user is not associated with this account

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

a.DoPurchase(5825, p2); //credit limit exceeded

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

c.Withdraw(1500, p5); //no overdraft

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

Bank.GetAccount("CK-100008"); //account does not exist

}

catch (AccountException e) { Console.WriteLine(e.Message); }

try

{

Bank.GetPerson("Trudeau"); //user does not exist

}

catch (AccountException e) { Console.WriteLine(e.Message); }

Console.WriteLine("\nBefore PrepareMonthlyReport()");

Console.WriteLine(c);

Console.WriteLine("\nAfter PrepareMonthlyReport()");

c.PrepareMonthlyReport(); //all transactions are cleared, balance changes

Console.WriteLine(c);