Programming II

# Assignment 1 – Implementing a Banking Application

This application will be implemented as a library. To verify your implementation, you will need to create a Console App. If this runs ok then you will move towards a GUI Application.

## Due: See course shell for due date and time

Graphical user interface

Description automatically generatedCheck with you professor to see if you are permitted to work as a group.

The application is by far the most complex system that you have attempted so far. It consists of eleven classes, two structs, one interface and two enums linked in varying degrees of tightness. The Bank class is the main driver of the application. It has a collection of Accounts and Person that is initialize appropriately in the static constructor. You will implement the entire system in Visual Studio. A short description of each type with their members is given below.

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

No method should have a **Console.WriteLine()** unless you are explicitly told to do so.

Each type must be in separate files.

## ExceptionType enum

You will implement this enum in Visual Studio. This type is used when creating exceptions is this application. There are seven members:

Copy the constants directly into Visual Studio

|  |
| --- |
| **ExceptionType**  enum |
| **Constants** |
| ACCOUNT\_DOES\_NOT\_EXIST,  CREDIT\_LIMIT\_HAS\_BEEN\_EXCEEDED,  NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT,  NO\_OVERDRAFT\_FOR\_THIS\_ACCOUNT,  PASSWORD\_INCORRECT,  USER\_DOES\_NOT\_EXIST,  USER\_NOT\_LOGGED\_IN |

The members are self-explanatory.

## AccountType enum

You will implement this enum in Visual Studio. There are four members:

|  |
| --- |
| **AccountType**  enum |
| **Constants** |
| Checking,  Saving,  Visa  Line\_of\_credit |

The members are self-explanatory.

## ITransaction interface

You will implement the ITransaction Interface in Visual Studio. Two of the sub classes of the Account class (CheckingAccount and SavingAccount) implement this interface.

A short description of each class member is given below:

|  |
| --- |
| **ITransaction**  Interface |
| **Properties** |
|  |
| **Methods** |
| Withdraw(amount : decimal, person : Person) : void  Deposit(amount : decimal, person : Person) : void |

### Properties:

There are no properties.

### Methods:

Remember all interfaces are by default abstract and public, hence there are no accessibility modifier such as public, protected or private. They are implicitly public!

1. void Withdraw(decimal amount, Person person).
2. void Deposit(decimal amount, Person person).

## 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 a single a constructor:

|  |
| --- |
| **AccountException**  Class  → Exception |
| **Fields** |
|  |
| **Properties** |
|  |
| **Methods** |
| + «Constructor» AccountException(reason : ExceptionType) |

### Fields:

There are no fields

### Properties:

There are no properties

### Constructor:

1. public AccountException(ExceptionType reason )– this public constructor simply invokes the base constructor with an appropriate argument. The base constructer argument is obtain by calling the ToString() of the argument. [You are sending the argument of the child constructor to the parent constructor.]

### Methods:

There are no methods.

## LoginEventArgs class

You will implement the **LoginEventArgs** Class in Visual Studio. This inherits from the **EventArgs** Class to provide a custom object for this application. It consists of two properties and a single a constructor:

|  |
| --- |
| **LoginEventArgs**  Class  → EventArgs |
| **Fields** |
|  |
| **Properties** |
| + «prop, no setter» PersonName : string  + «prop, no setter» Success : bool |
| **Methods** |
| + «Constructor» LoginEventArgs(  personName : string,  success : bool) |

### Fields:

There are no fields

### Properties:

There are two public properties, with no accompanying setters.

### Constructors:

1. public LoginEventArgs(string name, bool success) – this public constructor takes a string and a bool argument and does the following:
   1. Invokes the base constructor
   2. Assigns the arguments to the appropriate properties.

### Methods:

There are no explicitly defined methods.

## TransactionEventArgs class

You will implement the **TransactionEventArgs** Class in Visual Studio. This inherits from the **LoginEventArgs** Class to provide a custom object for this application. It consists of a single property and a single a constructor:

|  |
| --- |
| **TransactionEventArgs**  Class  → LoginEventArgs |
| **Fields** |
|  |
| **Properties** |
| + «prop, no set» Amount : decimal |
| **Methods** |
| + «Constructor» TransactionEventArgs(  personName : string,  amount : decimal,  success : bool) |

### Fields:

There are no fields

### Properties:

There is one explicitly defined public property, with no accompanying setters.

### Constructors:

1. public TransactionEventArgs(string name, decimal amount, bool success) – this public constructor takes a string, a double and a bool argument and does the following:
   1. Invokes the base constructor with the appropriate arguments.
   2. Assigns the middle argument to the appropriate property.

### Methods:

There are no explicitly defined methods.

## Transaction struct

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

|  |
| --- |
| **Transaction**  struct |
| **Fields** |
|  |
| **Properties** |
| + «prop, no set» AccountNumber : string  + «prop, no set» Amount : decimal  + «prop, no set» Originator : Person  + «prop, no set» Time : DayTime |
| **Methods** |
| + «Constructor» Transaction(  accountNumber : string,  amount : decimal,  ~~endBalance : double,~~  person : Person,  time : DayTime)  + ToString() : string |

### Properties:

There are no fields.

### Properties:

All the properties are public with the setter absent

1. AccountNumber – this property is a string representing the account number associated with this transaction. The getter is public, and the setter is absent.
2. Amount – this property is a decimal representing the account of this transaction. The getter is public, and the setter is absent.
3. Originator – this property is a Person representing the person initiating this transaction. The getter is public, and the setter is absent.
4. Time – this property is a DayTime type representing the time associated with this transaction. The getter is public, and the setter is absent.

### Constructors:

1. public Transaction( string accountNumber, decimal amount, Person person, DayTime time ) – This public constructor takes four arguments. It assigns the arguments to the appropriate properties.

### Methods:

1. 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. [See the output for clues for this method.] You must include the word Deposit (Amount is positive) or Withdraw (Amount is negative) in the output.

## DayTime struct

You will implement the DayTime struct in Visual Studio. The only purpose of this struct is to capture the time of an event.

The logic of this class depends on the following:

* 1 hour = 60 minutes
* 1 day = 24 hours = 24 \* 60 = 1\_440 minutes
* 1 month = 30 days = 30 \* 1\_440 = 43\_200 minutes
* 1 year = 12 months = 12 \* 43\_200 = 518\_400 minutes
* Time starts on the zero minute of the zero hour of the first day of the first month of the zero year. i.e. 0 minute will be 2023–01–01 00:00

A short description of the struct members is given below:

|  |
| --- |
| **DayTime**  struct |
| **Fields** |
| - minutes : long |
| **Properties** |
|  |
| **Methods** |
| + «Constructor» DayTime(minutes : long)  $+ «operator» +(  lhs: DayTime,  minute : int) : DayTime  + ToString() : string |

### Properties:

There is a single private field.

1. minutes – this private field is a long representing the minute value of this struct.  
   [The time on a computer system is a simple counter. The long counter represents the number of ticks (milliseconds) since 1970 January 1].   
   The minutes field will represent the number of minutes since the start of the year 2023.

### Properties:

There are no properties.

### Constructors:

1. public Day( long minutes ) – This public constructor takes a single long argument and assigns it to the appropriate field.

### Methods:

1. public static DayTime operator +( DayTime lhs, int minutes ) – This method overload the addition operator. Returns a new struct with the original minutes that is increased by the second argument. This is used in the Util class to get the current time (Utils.Now) and incremented time (Utils.Time).
2. public override string ToString( ) – This method overrides the same method of the Object class. It calculates the year, month, day, hour, and remaining minutes from the private minutes field. [See the output for clues for this method.]

## Logger class

You will implement the Logger Class in Visual Studio. This is a static class with some housekeeping duties. It handles all login attempts and transaction occurrences.

A short description of each class member is given below:

|  |
| --- |
| **Logger**  Static Class |
| **Fields** |
| $- loginEvents : List<string>  $- transactionEvents : List<string> |
| **Properties** |
|  |
| **Methods** |
| $+ «EventHandler» LoginHandler(  sender : object,  args : EventArgs) : void  $+ «EventHandler» TransactionHandler(  sender : object,  args : EventArgs) : void  $+ SaveLoginEvents(filename : **string**) : void  $+ SaveTransactionEvents(filename : **string**) : void |

### Fields:

There are two fields:

1. loginEvents – this private static List<string>field represents login attempts to the bank. This is initialized to an empty collection at declaration.
2. transactionEvents – this private static List<string>field represents login attempts to the bank. This is initialized to an empty collection at declaration.

### Properties:

There are no properties.

### Constructors:

There are no properties.

### Methods:

1. **public static void LoginHandler(object sender, EventArgs args)** – This method does the following:
   1. Uses the **as** operator to cast the second argument to a **LoginEventArgs** object.
   2. Create a string with the following:

* PersonName (from args).
* Success (from args).
* Current time (from **Utils.Now**).
  1. Adds the above string to the loginEvents collection.

This method does not display anything

1. **public static void TransactionHandler(object sender, EventArgs args)** – This method does the following:
   1. Uses the **as** operator to cast the second argument to a **TransactionEventArgs** object.
   2. Create a string with the following:

* PersonName (from args).
* Amount (from args)
* Operation (from Amount in args)
* Success (from args).
* Current time (from **Utils.Now**).
  1. Adds the above string to the transactionEvents collection.

This method does not display anything

1. **public static void ShowLoginEvents(**filename : **string )** – This is method does not take any parameters nor does it return a value. The method does the following:
   1. Writes the current time (**Utils.Now**) to the file.
   2. Write a numbered list of all the items in the **loginEvents** collection. To the file
2. **public static void ShowTransactionEvents( )** – This is method does not take any parameters nor does it return a value. The method does the following:
   1. Prints the current time (**Utils.Now**)
   2. Prints a number list of all the items in the **transactionEvents** collection.

## Person class

You will implement the Person Class in Visual Studio. A person object may be associated with multiple accounts. A person initiates activities (deposits or withdrawal) against an account that is captured in a transaction object.

A short description of each class member is given below:

|  |
| --- |
| **Person**  Class |
| **Fields** |
| - password : string  + OnLogin : event Eventhandler |
| **Properties** |
| + «C# property, no set » Sin : string  + «C# property, no set » Name : string  + «C# property, private set» IsAuthenticated : bool |
| **Methods** |
| + «Constructor» Person(  name : string,  sin : string)  + Login(password : string) : void  + Logout() : void  + ToString() : string |

### Fields:

There are two fields:

1. password – this private 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 resulting hash value is stored. When a user supplies a password, it is passed through the same hashing function and the result is compared to the stored value.)
2. OnLogin – this public event field represents the EventHandler that will store the method to be invoked.

### Properties:

All the properties are auto-implemented.

1. SIN – this string property represents the sin number of this person. The getter is public, and setter is absent.
2. Name – this property is a string representing the name of this person. This getter is public, and setter is absent.
3. 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 getter is public, and setter is absent.

### Constructors:

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 field to the first three letters of the SIN.  
      [use the Substring(start\_position, length) method of the string class]

### Methods:

1. public void Login(string password) – This method takes a string parameter representing a password and does the following:
   1. If the argument DOES NOT match the password field, it does the following:

* Sets the IsAuthenticated property to false
* Invokes the EventHandler OnLogin with the following arguments:
  + The current object reference (use this)
  + The second argument is of type LoginEventArgs. This will have arguments
    - the name of the current object
    - and the success on the operation (use false in this case).
* Creates an AccountException object using argument ExceptionType.PASSWORD\_INCORRECT
* Throws the above exception
  1. If the argument matches the password, it does the following:
* Sets the IsAuthenticated property is set to true
* Invokes the EventHandler OnLogin with the following arguments:
  + The current object reference (use this)
  + The second argument is of type LoginEventArgs. This will have arguments
    - the name of the current object
    - and the success on the operation (use true in this case).

This method does not display anything

1. public void Logout( ) – This is public method does not take any parameters nor does it return a value. It sets the IsAuthenticated property to false.
2. public override string ToString( )– This public method overrides the same method of the Object class. It returns a string representing the name of the person ~~and if he is authenticated or not~~.

## Account class

You will implement the Account Class in Visual Studio. This class that will serve as the base class for the Visa, Checking and Saving classes. The member PrepareMonthlyStatement() is abstract so the class must be declared abstract. 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** |
| $- LAST\_NUMBER = 100\_000 : int  # «readonly» users : List<Person>  + «readonly» transactions : List<Transaction>  + «virtual»OnLogin : event Eventhandler  + «virtual»OnTransaction : event Eventhandler |
| **Properties** |
| + «prop, no set» Number : string  + «prop, protected set» Balance : decimal  + «prop, protected set» LowestBalance : decimal |
| **Methods** |
| + «Constructor» Account(type : string, balance : decimal)  + Deposit(balance : decimal, person : Person) : void  + AddUser(Person : Person) : void  + IsUser(name : string) : bool  + «virtual» OnTransactionOccur(  Sender : object  args :EventArgs) : void  + «abstract» PrepareMonthlyStatement(): void  + ToString() : string |

### Fields:

If these two fields are readonly, then how it is possible to add users and transactions?

1. users – this readonly protected field is a list of persons representing the user who have access to this account. This is initialized at in the constructor to an empty list of Person at declaration.
2. transactions – this readonly public field is a list of transaction representing the deposits, withdrawal, payments and purchases of this account. This initialized at declaration to an empty list of Transaction.
3. 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. It is initialized at declaration to 100,000 and is modified in the constructor.

### 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.
3. Number – this property is a string representing the account number of this account. The getter is public and the setter is absent.

### Methods:

1. public Account( string type, decimal balance ) – This is the public constructor. It takes two parameters: a three-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 does the following:
   1. Sets the Number property to the concatenation (joining) of the first argument and the class variable LastNumber.
   2. Increments the class variable LastNumber.
   3. Assigns the second argument to the property Balance.
   4. Assigns the second argument to the property LowestBalance.

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

1. public void Deposit( decimal 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. *(We are not checking if the person argument is authenticated or not because it is not meant to be called directly but tot be called by the child classes of this class.)* This method does the following:
   1. Increase (or decrease) the property Balance by the amount specified by its argument.
   2. Update the property LowestBalance based on the current value of Balance.
   3. Create a Transaction object based on the account Number, current time (use Utils.Now), the amount (specified by the argument), a person object ( specified by the argument.
   4. Adds the above object to the list of transactions

This is method is called by the Deposit and Withdraw methods of the derived classes CheckingAccount and SavingsAccount 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 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 bool IsUser( 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.  
   This method does not display anything on screen

**Note**: This is not a sound design because it is possible that multiple users will have the same name

1. *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.*
2. public virtual void OnTransactionOccur(object sender, EventArgs e) – This public method invoke the EventHandler OnTransaction with its two arguments. This method does not return a value, nor does it display anything on the screen.
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:
      * The Number of this account
      * The names of each of the users of the account
      * The Balance
      * All the transactions

## CheckingAccount class

You will implement the CheckingAccount Class in Visual Studio. This class inherits from the Account class and implements the ITransaction interface. There are two class variables i.e. variables that are shared but all the objects of this class. A short description of the class members is given below:

|  |
| --- |
| **CheckingAccount**  Class  → Account, ITransaction |
| **Fields** |
| $- COST\_PER\_TRANSACTION = 0.05 : decimal  $- INTEREST\_RATE = 0.005 : decimal  - hasOverdraft : bool |
| **Methods** |
| + «Constructor» CheckingAccount(balance = 0 : decimal, hasOverdraft = false: bool)  + Deposit(amount : decimal, person : Person) : void  + Withdraw(amount : decimal, 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 0.05m. The m suffix is necessary when working with decimal literals.
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 0.005m.
3. hasOverdraft – this is a bool indicating if the balance on this account can be less than zero. This private instance (instance means non-static) variable is set in the constructor.

### Methods:

1. public CheckingAccount( decimal 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 appropriate argument.
   2. Assigns the hasOverdraft argument to the appropriate field.
2. public new void Deposit( decimal 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:

This definition hides the corresponding member in the parent class because the base class implementation is needed for this method as well as the Withdraw() method

* 1. Calls the Deposit() method of the base class with the appropriate arguments.
  2. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
     1. The second argument TransactionEventArgs needs the name of the person, the amount and true (success of the operation) OnTransactionOccur

1. public void Withdraw( decimal 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. If this person in not associated with this account, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments. Read the above Deposit() method for more explanation.
      2. Throws the appropriate AccountException object
   2. If this person in not logged in, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. Throws the appropriate AccountException object.
   3. If the withdrawal amount is greater than the balance and there is no overdraft facility, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. Throws the appropriate AccountException object.
   4. Otherwise, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. calls the Deposit() method of this base class with the appropriate arguments (you will send negative of the amount).
2. 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. Re-initializes the collection transactions (use the Clear() method of the list class)

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

In a real-world application, the transaction objects would be archived before clearing.

## SavingAccount class

You will implement the SavingAccount Class in Visual Studio. This is a sub class derived from the Account class and implements the ITransaction interface. Again, there are two class variables. A short description of the class members is given below:

|  |
| --- |
| **SavingAccount**  Class  → Account, ITransaction |
| **Fields** |
| $- COST\_PER\_TRANSACTION = 0.5 : decimal  $- INTEREST\_RATE = 0.015 : decimal |
| **Methods** |
| + «Constructor» SavingAccount(  balance = 0 : decimal)  + Deposit(  amount : decimal,  person : Person) : void  + Withdraw(  amount : decimal,  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( decimal balance = 0 ) – This public constructor takes a parameter of type double representing the starting balance of the account. The constructor does the following:
   1. It invokes the base constructor with the string “SV-” and its argument.
2. public new void Deposit( decimal 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. This method does the following:
   1. Calls the Deposit() method of the base class with the appropriate arguments.
   2. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      1. The second argument TransactionEventArgs needs the name of the person, the amount and true (success of the operation)
3. public void Withdraw( decimal 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. If this person in not associated with this account, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments. Read the above Deposit() method for more explanation
      2. Throws the appropriate AccountException object
   2. If this person in not logged in, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. Throws the appropriate AccountException object.
   3. If the withdrawal amount is greater than the balance and there is no overdraft facility, it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. Throws the appropriate AccountException object.
   4. Otherwise it does the following:
      1. Calls the OnTransactionOccur() method of the base class with the appropriate arguments.
      2. 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 parameters, nor does it display anything

## VisaAccount class

You will implement the VisaAccount Class in Visual Studio. This is a sub class derived from the **Account** class and implements **ITransaction** interface. This class has a single class variable. A short description of the class members is given below:

|  |
| --- |
| **VisaAccount**  Class  → Account, ITransaction |
| **Fields** |
| - creditLimit : decimal  $- INTEREST\_RATE = 0.1995 : decimal |
| **Methods** |
| + «Constructor» VisaAccount(  balance = 0 : **double**,  creditLimit = 1200: decimal)  + DoPayment(  amount : decimal,  person : **Person**) : **void**  + DoPurchase(  amount : decimal,  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 **0.1995m**.

### Methods:

1. **public VisaAccount(** decimal **balance = 0,** decimal **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 appropriate argument.
   2. Assigns the argument to the appropriate field
2. **public void DoPayment(** decimal **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. This method does the following:
   1. Calls the **Deposit()** method of the base class with the appropriate arguments.
   2. Calls the **OnTransactionOccur()** method of the base class with the appropriate arguments.
      1. The second argument TransactionEventArgs needs the name of the person, the amount and true (success of the operation).
3. **public void DoPurchase(** decimal **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. If this person in not associated with this account, it does the following:
      1. Calls the **OnTransactionOccur()** method of the base class with the appropriate arguments. Read the above Deposit() method for more explanation
      2. Throws the appropriate **AccountException** object
   2. If this person in not logged in, it does the following:
      1. Calls the **OnTransactionOccur()** method of the base class with the appropriate arguments.
      2. Throws the appropriate **AccountException** object.
   3. If the purchase amount is greater than the sum of the balance and the credit limit, it does the following:
      1. Calls the **OnTransactionOccur()** method of the base class with the appropriate arguments.
      2. Throws the appropriate **AccountException** object.
   4. Otherwise it does the following:
      1. Calls the **OnTransactionOccur()** method of the base class with the appropriate arguments.
      2. 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:

|  |
| --- |
| **Bank**  Static Class |
| **Fields** |
| $+ «readonly» ACCOUNTS : **Dictionary**<**string**, **Account**>  $+ «readonly» USERS : **Dictionary**<**string**, **Person**> |
| **Methods** |
| $ «Constructor» Bank()  $+ AddUser(name : **string**, sin : **string**) : **void**  $+ AddAccount(account : **Account**) : **void**  $+ AddUserToAccount(number : **string**, name : **string**) : **void**  $+ GetAccount(number : **string**) : **Account**  $+ GetUser(name : **string**) : **Person**  $+ SaveAccounts(filename : **string**) : **void**  $+ SaveUsers(filename : **string**) : **void**  $+ GetAllTransactions() : **List**<**Transaction**> |

### Fields:

1. **ACCOUNTS** – this public readonly class variable is a dictionary of account. It is initialized at declaration.
2. **USERS** – this public readonly class variable is a dictionary of person. It is initialized at declaration.

### Methods:

1. **static Bank( )** – This static constructor contains the following statements to initialize the USERS and ACCOUNTS collections.

//initialize the USERS collection

AddPerson("Narendra", "1234-5678"); //0

AddPerson("Ilia", "2345-6789"); //1

AddPerson("Mehrdad", "3456-7890"); //2

AddPerson("Vinay", "4567-8901"); //3

AddPerson("Arben", "5678-9012"); //4

AddPerson("Patrick", "6789-0123"); //5

AddPerson("Yin", "7890-1234"); //6

AddPerson("Hao", "8901-2345"); //7

AddPerson("Jake", "9012-3456"); //8

AddPerson("Mayy", "1224-5678"); //9

AddPerson("Nicoletta", "2344-6789"); //10

//initialize the ACCOUNTS collection

AddAccount(new VisaAccount()); //VS-100000

AddAccount(new VisaAccount(150, -500)); //VS-100001

AddAccount(new SavingAccount(5000)); //SV-100002

AddAccount(new SavingAccount()); //SV-100003

AddAccount(new CheckingAccount(2000)); //CK-100004

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

AddAccount(new VisaAccount(50, -550)); //VS-100006

AddAccount(new SavingAccount(1000)); //SV-100007

//associate users with accounts

string number = "VS-100000";

AddUserToAccount(number, "Narendra");

AddUserToAccount(number, "Ilia");

AddUserToAccount(number, "Mehrdad");

number = "VS-100001";

AddUserToAccount(number, "Vinay");

AddUserToAccount(number, "Arben");

AddUserToAccount(number, "Patrick");

number = "SV-100002";

AddUserToAccount(number, "Yin");

AddUserToAccount(number, "Hao");

AddUserToAccount(number, "Jake");

number = "SV-100003";

AddUserToAccount(number, "Mayy");

AddUserToAccount(number, "Nicoletta")

number = "CK-100004";

AddUserToAccount(number, "Mehrdad");

AddUserToAccount(number, "Arben");

AddUserToAccount(number, "Yin");

number = "CK-100005";

AddUserToAccount(number, "Jake");

AddUserToAccount(number, "Nicoletta")

number = "VS-100006";

AddUserToAccount(number, "Ilia");

AddUserToAccount(number, "Vinay");

number = "SV-100007";

AddUserToAccount(number, "Patrick");

AddUserToAccount(number, "Hao");

1. **public static void SaveAccounts( string filename)** – this public static method serialize all the accounts in the ACCOUNTS collection to the appropriate file in json format.
2. **public static void SaveUsers( string filename)** – this public static method serializes all the persons in the USERS collection to the appropriate file in json format.
3. **public static Person GetUser( string name )** – this public static method takes a string representing the name of a person and returns the matching person object. This method does the following:
   1. Check if the argument is present in the key of the static collection USERS.
   2. If the argument is present the value part of this pair is returned.
   3. Else the appropriate **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. This method does the following:
   1. Check if the argument is present in the key of the static collection ACCOUNTS.
   2. If the argument is present the value part of this pair is returned.
   3. Else the appropriate **AccountException** object is thrown
2. **public static void AddUser(string name, string sin)** – this method takes two strings and does the following:
3. Creates a Person object with the two arguments.
4. Add the static method **LoginHandler()** of the **Logger** class to the **Eventhandler** field (OnLogin) of the above object.
5. Adds a key-value pair to the USERS dictionary. The key is the second argument of the method and the value is the object created in step a.
6. **public static void AddAccount(Account account)** – this method takes an account argument and does the following:
7. Add the static method **TransactionHandler()** of the **Logger** class to the **Eventhandler** field (OnTransaction) of the argument.
8. Adds a key-value pair to the ACCOUNTS collection. The key is the Number property of the argument and the value is the actual argument respectively.
9. **public static void AddUserToAccount(string number, string name)** – this method takes two string arguments and does the following:
10. Locates the account matching the first argument.
11. Locates the person matching the second argument.
12. Invokes the AddUser() method on the account object and passing the person object.

## Testing

Use the following code in your test harness.

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

Bank.PrintAccounts();

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

Bank.PrintPersons();

Person p0, p1, p2, p3, p4, p5, p6, p7, p8, p9, p10;

p0 = Bank.GetPerson("Narendra");

p1 = Bank.GetPerson("Ilia");

p2 = Bank.GetPerson("Mehrdad");

p3 = Bank.GetPerson("Vinay");

p4 = Bank.GetPerson("Arben");

p5 = Bank.GetPerson("Patrick");

p6 = Bank.GetPerson("Yin");

p7 = Bank.GetPerson("Hao");

p8 = Bank.GetPerson("Jake");

p9 = Bank.GetPerson("Mayy");

p10 = Bank.GetPerson("Nicoletta");

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

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

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

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

p10.Login("234"); p8.Login("901");

//a visa account

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

a.Pay(1500, p0);

a.Purchase(200, p1);

a.Purchase(25, p2);

a.Purchase(15, p0);

a.Purchase(39, p1);

a.Pay(400, p0);

Console.WriteLine(a);

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

a.Pay(500, p0);

a.Purchase(25, p3);

a.Purchase(20, p4);

a.Purchase(15, p5);

Console.WriteLine(a);

//a saving account

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

b.Withdraw(300, p6);

b.Withdraw(32.90, p6);

b.Withdraw(50, p7);

b.Withdraw(111.11, p8);

Console.WriteLine(b);

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

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, p10);

Console.WriteLine(b);

//a checking account

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

c.Deposit(33.33, p7);

c.Deposit(40.44, p7);

c.Withdraw(150, p2);

c.Withdraw(200, p4);

c.Withdraw(645, p6);

c.Withdraw(350, p6);

Console.WriteLine(c);

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

c.Deposit(33.33, p8);

c.Deposit(40.44, p7);

c.Withdraw(450, p10);

c.Withdraw(500, p8);

c.Withdraw(645, p10);

c.Withdraw(850, p10);

Console.WriteLine(c);

a = Bank.GetAccount("VS-100006") as VisaAccount;

a.Pay(700, p0);

a.Purchase(20, p3);

a.Purchase(10, p1);

a.Purchase(15, p1);

Console.WriteLine(a);

b = Bank.GetAccount("SV-100007") as SavingAccount;

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, p7);

Console.WriteLine(b);

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

}

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

try

{

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

}

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

try

{

a.Purchase(5825, p4); //credit limit exceeded

}

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

try

{

c.Withdraw(1500, p6); //no overdraft

}

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

try

{

Bank.GetAccount("CK-100018"); //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); }

//show all transactions

Console.WriteLine("\n\nAll transactions");

foreach(var transaction in Bank.GetAllTransactions())

Console.WriteLine( transaction);

foreach (var keyValuePair in Bank.ACCOUNTS)

{

Account account = keyValuePair.Value;

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

Console.WriteLine(account);

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

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

Console.WriteLine(account);

}

Logger.ShowLoginEvents();

Logger.ShowTransactionEvents();

## Sample output

Use the following code in your test harness.

All acounts:

[VS-100000, VS-100000 Narendra, Ilia, Mehrdad $0.00

]

[VS-100001, VS-100001 Vinay, Arben, Patrick $150.00

]

[SV-100002, SV-100002 Yin, Hao, Jake $5,000.00

]

[SV-100003, SV-100003 Mayy, Nicoletta $0.00

]

[CK-100004, CK-100004 Mehrdad, Arben, Yin $2,000.00

]

[CK-100005, CK-100005 Jake, Nicoletta $1,500.00

]

[VS-100006, VS-100006 Ilia, Vinay $50.00

]

[SV-100007, SV-100007 Patrick, Hao $1,000.00

]

All Users:

[Narendra, Narendra]

[Ilia, Ilia]

[Mehrdad, Mehrdad]

[Vinay, Vinay]

[Arben, Arben]

[Patrick, Patrick]

[Yin, Yin]

[Hao, Hao]

[Jake, Jake]

[Mayy, Mayy]

[Nicoletta, Nicoletta]

VS-100000 Narendra, Ilia, Mehrdad $1,621.00

VS-100000 $1,500.00 deposited by Narendra on 2021-08-12 16:52

VS-100000 $200.00 withdrawn by Ilia on 2021-08-13 10:08

VS-100000 $25.00 withdrawn by Mehrdad on 2021-08-14 04:59

VS-100000 $15.00 withdrawn by Narendra on 2021-08-15 01:05

VS-100000 $39.00 withdrawn by Ilia on 2021-08-16 01:07

VS-100000 $400.00 deposited by Narendra on 2021-08-16 18:04

VS-100001 Vinay, Arben, Patrick $590.00

VS-100001 $500.00 deposited by Narendra on 2021-08-17 10:00

VS-100001 $25.00 withdrawn by Vinay on 2021-08-18 09:57

VS-100001 $20.00 withdrawn by Arben on 2021-08-19 02:00

VS-100001 $15.00 withdrawn by Patrick on 2021-08-19 20:18

SV-100002 Yin, Hao, Jake $4,505.99

SV-100002 $300.00 withdrawn by Yin on 2021-08-19 22:58

SV-100002 $32.90 withdrawn by Yin on 2021-08-20 14:46

SV-100002 $50.00 withdrawn by Hao on 2021-08-21 10:30

SV-100002 $111.11 withdrawn by Jake on 2021-08-22 07:58

SV-100003 Mayy, Nicoletta $271.79

SV-100003 $300.00 deposited by Vinay on 2021-08-22 19:46

SV-100003 $32.90 deposited by Mehrdad on 2021-08-23 11:57

SV-100003 $50.00 deposited by Patrick on 2021-08-23 13:35

SV-100003 $111.11 withdrawn by Nicoletta on 2021-08-24 14:24

CK-100004 Mehrdad, Arben, Yin $728.77

CK-100004 $33.33 deposited by Hao on 2021-08-25 14:45

CK-100004 $40.44 deposited by Hao on 2021-08-26 05:09

CK-100004 $150.00 withdrawn by Mehrdad on 2021-08-26 21:59

CK-100004 $200.00 withdrawn by Arben on 2021-08-28 05:15

CK-100004 $645.00 withdrawn by Yin on 2021-08-28 22:37

CK-100004 $350.00 withdrawn by Yin on 2021-08-29 23:52

CK-100005 Jake, Nicoletta -$871.23

CK-100005 $33.33 deposited by Jake on 2021-08-30 21:35

CK-100005 $40.44 deposited by Hao on 2021-09-01 06:32

CK-100005 $450.00 withdrawn by Nicoletta on 2021-09-02 03:00

CK-100005 $500.00 withdrawn by Jake on 2021-09-02 10:10

CK-100005 $645.00 withdrawn by Nicoletta on 2021-09-02 17:43

CK-100005 $850.00 withdrawn by Nicoletta on 2021-09-03 10:00

VS-100006 Ilia, Vinay $705.00

VS-100006 $700.00 deposited by Narendra on 2021-09-04 00:15

VS-100006 $20.00 withdrawn by Vinay on 2021-09-04 18:57

VS-100006 $10.00 withdrawn by Ilia on 2021-09-05 23:26

VS-100006 $15.00 withdrawn by Ilia on 2021-09-07 03:11

SV-100007 Patrick, Hao $1,271.79

SV-100007 $300.00 deposited by Vinay on 2021-09-08 01:49

SV-100007 $32.90 deposited by Mehrdad on 2021-09-08 20:31

SV-100007 $50.00 deposited by Patrick on 2021-09-09 15:18

SV-100007 $111.11 withdrawn by Hao on 2021-09-09 20:26

Exceptions:

PASSWORD\_INCORRECT

USER\_NOT\_LOGGED\_IN

NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT

NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT

NAME\_NOT\_ASSOCIATED\_WITH\_ACCOUNT

ACCOUNT\_DOES\_NOT\_EXIST

USER\_DOES\_NOT\_EXIST

All transactions

VS-100000 $1,500.00 deposited by Narendra on 2021-08-12 16:52

VS-100000 $200.00 withdrawn by Ilia on 2021-08-13 10:08

VS-100000 $25.00 withdrawn by Mehrdad on 2021-08-14 04:59

VS-100000 $15.00 withdrawn by Narendra on 2021-08-15 01:05

VS-100000 $39.00 withdrawn by Ilia on 2021-08-16 01:07

VS-100000 $400.00 deposited by Narendra on 2021-08-16 18:04

VS-100001 $500.00 deposited by Narendra on 2021-08-17 10:00

VS-100001 $25.00 withdrawn by Vinay on 2021-08-18 09:57

VS-100001 $20.00 withdrawn by Arben on 2021-08-19 02:00

VS-100001 $15.00 withdrawn by Patrick on 2021-08-19 20:18

SV-100002 $300.00 withdrawn by Yin on 2021-08-19 22:58

SV-100002 $32.90 withdrawn by Yin on 2021-08-20 14:46

SV-100002 $50.00 withdrawn by Hao on 2021-08-21 10:30

SV-100002 $111.11 withdrawn by Jake on 2021-08-22 07:58

SV-100003 $300.00 deposited by Vinay on 2021-08-22 19:46

SV-100003 $32.90 deposited by Mehrdad on 2021-08-23 11:57

SV-100003 $50.00 deposited by Patrick on 2021-08-23 13:35

SV-100003 $111.11 withdrawn by Nicoletta on 2021-08-24 14:24

CK-100004 $33.33 deposited by Hao on 2021-08-25 14:45

CK-100004 $40.44 deposited by Hao on 2021-08-26 05:09

CK-100004 $150.00 withdrawn by Mehrdad on 2021-08-26 21:59

CK-100004 $200.00 withdrawn by Arben on 2021-08-28 05:15

CK-100004 $645.00 withdrawn by Yin on 2021-08-28 22:37

CK-100004 $350.00 withdrawn by Yin on 2021-08-29 23:52

CK-100005 $33.33 deposited by Jake on 2021-08-30 21:35

CK-100005 $40.44 deposited by Hao on 2021-09-01 06:32

CK-100005 $450.00 withdrawn by Nicoletta on 2021-09-02 03:00

CK-100005 $500.00 withdrawn by Jake on 2021-09-02 10:10

CK-100005 $645.00 withdrawn by Nicoletta on 2021-09-02 17:43

CK-100005 $850.00 withdrawn by Nicoletta on 2021-09-03 10:00

VS-100006 $700.00 deposited by Narendra on 2021-09-04 00:15

VS-100006 $20.00 withdrawn by Vinay on 2021-09-04 18:57

VS-100006 $10.00 withdrawn by Ilia on 2021-09-05 23:26

VS-100006 $15.00 withdrawn by Ilia on 2021-09-07 03:11

SV-100007 $300.00 deposited by Vinay on 2021-09-08 01:49

SV-100007 $32.90 deposited by Mehrdad on 2021-09-08 20:31

SV-100007 $50.00 deposited by Patrick on 2021-09-09 15:18

SV-100007 $111.11 withdrawn by Hao on 2021-09-09 20:26

Before PrepareMonthlyReport()

VS-100000 Narendra, Ilia, Mehrdad $1,621.00

VS-100000 $1,500.00 deposited by Narendra on 2021-08-12 16:52

VS-100000 $200.00 withdrawn by Ilia on 2021-08-13 10:08

VS-100000 $25.00 withdrawn by Mehrdad on 2021-08-14 04:59

VS-100000 $15.00 withdrawn by Narendra on 2021-08-15 01:05

VS-100000 $39.00 withdrawn by Ilia on 2021-08-16 01:07

VS-100000 $400.00 deposited by Narendra on 2021-08-16 18:04

After PrepareMonthlyReport()

VS-100000 Narendra, Ilia, Mehrdad $1,621.00

Before PrepareMonthlyReport()

VS-100001 Vinay, Arben, Patrick $590.00

VS-100001 $500.00 deposited by Narendra on 2021-08-17 10:00

VS-100001 $25.00 withdrawn by Vinay on 2021-08-18 09:57

VS-100001 $20.00 withdrawn by Arben on 2021-08-19 02:00

VS-100001 $15.00 withdrawn by Patrick on 2021-08-19 20:18

After PrepareMonthlyReport()

VS-100001 Vinay, Arben, Patrick $619.93

Before PrepareMonthlyReport()

SV-100002 Yin, Hao, Jake $4,505.99

SV-100002 $300.00 withdrawn by Yin on 2021-08-19 22:58

SV-100002 $32.90 withdrawn by Yin on 2021-08-20 14:46

SV-100002 $50.00 withdrawn by Hao on 2021-08-21 10:30

SV-100002 $111.11 withdrawn by Jake on 2021-08-22 07:58

After PrepareMonthlyReport()

SV-100002 Yin, Hao, Jake $4,573.38

Before PrepareMonthlyReport()

SV-100003 Mayy, Nicoletta $271.79

SV-100003 $300.00 deposited by Vinay on 2021-08-22 19:46

SV-100003 $32.90 deposited by Mehrdad on 2021-08-23 11:57

SV-100003 $50.00 deposited by Patrick on 2021-08-23 13:35

SV-100003 $111.11 withdrawn by Nicoletta on 2021-08-24 14:24

After PrepareMonthlyReport()

SV-100003 Mayy, Nicoletta $271.59

Before PrepareMonthlyReport()

CK-100004 Mehrdad, Arben, Yin $728.77

CK-100004 $33.33 deposited by Hao on 2021-08-25 14:45

CK-100004 $40.44 deposited by Hao on 2021-08-26 05:09

CK-100004 $150.00 withdrawn by Mehrdad on 2021-08-26 21:59

CK-100004 $200.00 withdrawn by Arben on 2021-08-28 05:15

CK-100004 $645.00 withdrawn by Yin on 2021-08-28 22:37

CK-100004 $350.00 withdrawn by Yin on 2021-08-29 23:52

After PrepareMonthlyReport()

CK-100004 Mehrdad, Arben, Yin $732.11

Before PrepareMonthlyReport()

CK-100005 Jake, Nicoletta -$871.23

CK-100005 $33.33 deposited by Jake on 2021-08-30 21:35

CK-100005 $40.44 deposited by Hao on 2021-09-01 06:32

CK-100005 $450.00 withdrawn by Nicoletta on 2021-09-02 03:00

CK-100005 $500.00 withdrawn by Jake on 2021-09-02 10:10

CK-100005 $645.00 withdrawn by Nicoletta on 2021-09-02 17:43

CK-100005 $850.00 withdrawn by Nicoletta on 2021-09-03 10:00

After PrepareMonthlyReport()

CK-100005 Jake, Nicoletta -$875.89

Before PrepareMonthlyReport()

VS-100006 Ilia, Vinay $705.00

VS-100006 $700.00 deposited by Narendra on 2021-09-04 00:15

VS-100006 $20.00 withdrawn by Vinay on 2021-09-04 18:57

VS-100006 $10.00 withdrawn by Ilia on 2021-09-05 23:26

VS-100006 $15.00 withdrawn by Ilia on 2021-09-07 03:11

After PrepareMonthlyReport()

VS-100006 Ilia, Vinay $714.98

Before PrepareMonthlyReport()

SV-100007 Patrick, Hao $1,271.79

SV-100007 $300.00 deposited by Vinay on 2021-09-08 01:49

SV-100007 $32.90 deposited by Mehrdad on 2021-09-08 20:31

SV-100007 $50.00 deposited by Patrick on 2021-09-09 15:18

SV-100007 $111.11 withdrawn by Hao on 2021-09-09 20:26

After PrepareMonthlyReport()

SV-100007 Patrick, Hao $1,286.59

Login events as of 2021-09-10 22:31

1 Narendra logged in successfully on 2021-08-09 06:48

2 Ilia logged in successfully on 2021-08-09 15:26

3 Mehrdad logged in successfully on 2021-08-10 07:19

4 Vinay logged in successfully on 2021-08-10 17:49

5 Arben logged in successfully on 2021-08-10 19:41

6 Patrick logged in successfully on 2021-08-11 10:53

7 Yin logged in successfully on 2021-08-11 20:42

8 Hao logged in successfully on 2021-08-11 23:33

9 Nicoletta logged in successfully on 2021-08-12 02:09

10 Jake logged in successfully on 2021-08-12 09:50

11 Jake logged in unsuccessfully on 2021-09-09 21:26

Transaction events as of 2021-09-10 22:31

1 Narendra deposit $1,500.00 successfully on 2021-08-12 11:59

2 Ilia deposit $200.00 successfully on 2021-08-12 18:42

3 Mehrdad deposit $25.00 successfully on 2021-08-13 14:04

4 Narendra deposit $15.00 successfully on 2021-08-14 09:07

5 Ilia deposit $39.00 successfully on 2021-08-15 15:37

6 Narendra deposit $400.00 successfully on 2021-08-16 02:57

7 Narendra deposit $500.00 successfully on 2021-08-17 08:20

8 Vinay deposit $25.00 successfully on 2021-08-17 22:27

9 Arben deposit $20.00 successfully on 2021-08-19 00:51

10 Patrick deposit $15.00 successfully on 2021-08-19 17:22

11 Yin deposit $300.00 successfully on 2021-08-19 22:42

12 Yin deposit $32.90 successfully on 2021-08-20 02:00

13 Hao deposit $50.00 successfully on 2021-08-21 06:57

14 Jake deposit $111.11 successfully on 2021-08-21 16:10

15 Vinay deposit $300.00 successfully on 2021-08-22 12:26

16 Mehrdad deposit $32.90 successfully on 2021-08-23 07:12

17 Patrick deposit $50.00 successfully on 2021-08-23 13:35

18 Nicoletta deposit $111.11 successfully on 2021-08-24 00:43

19 Hao deposit $33.33 successfully on 2021-08-25 03:29

20 Hao deposit $40.44 successfully on 2021-08-25 21:00

21 Mehrdad deposit $150.00 successfully on 2021-08-26 06:32

22 Arben deposit $200.00 successfully on 2021-08-27 13:54

23 Yin deposit $645.00 successfully on 2021-08-28 08:57

24 Yin deposit $350.00 successfully on 2021-08-29 09:39

25 Jake deposit $33.33 successfully on 2021-08-30 14:34

26 Hao deposit $40.44 successfully on 2021-08-30 21:53

27 Nicoletta deposit $450.00 successfully on 2021-09-01 10:40

28 Jake deposit $500.00 successfully on 2021-09-02 08:17

29 Nicoletta deposit $645.00 successfully on 2021-09-02 12:00

30 Nicoletta deposit $850.00 successfully on 2021-09-03 09:37

31 Narendra deposit $700.00 successfully on 2021-09-03 19:15

32 Vinay deposit $20.00 successfully on 2021-09-04 11:45

33 Ilia deposit $10.00 successfully on 2021-09-05 09:41

34 Ilia deposit $15.00 successfully on 2021-09-06 12:55

35 Vinay deposit $300.00 successfully on 2021-09-07 10:23

36 Mehrdad deposit $32.90 successfully on 2021-09-08 16:54

37 Patrick deposit $50.00 successfully on 2021-09-09 08:05

38 Hao deposit $111.11 successfully on 2021-09-09 16:50

39 Vinay deposit $12.50 unsuccessfully on 2021-09-10 01:47

40 Narendra deposit $12.50 unsuccessfully on 2021-09-10 10:50

41 Arben deposit $5,825.00 unsuccessfully on 2021-09-10 13:21

42 Yin deposit $1,500.00 unsuccessfully on 2021-09-10 22:31