**C# OOP Exam Regular – 5 August 2023**



**Bank Loan**

* **Overview**

*We are a new bank committed to providing personalized financial services to our customers. Our primary focus is on building strong relationships and understanding the unique needs of each client. We specialize in offering competitive loans to individuals and businesses, with flexible terms and competitive interest rates. Our streamlined application process ensures a hassle-free experience for our customers. With a dedicated team of professionals, we aim to deliver exceptional service and support to help our clients achieve their financial goals.*

* **Setup**
* Upload **only the BankLoan** project in every task **except** **Unit Tests.**
* **Do not modify the interfaces or their packages.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces wherever possible**:
* This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.
* **Exception messages** and **output messages** can be found in the **"Utilities"** folder.
* For solving this problem use **Visual Studio 2019, Visual Studio 2022** and **netcoreapp 3.1, netcoreapp 6.0**
* **NOTE: Do not use** "\r\n" **for a new line.**
* **Task 1: Structure (50 points)**

**For this task’s evaluation logic in the methods isn’t included.**

You are given some interfaces, and you have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Loan, Client** and **Bank**.

There should also be a **LoanRepository** and a **BankRepository**, implementing the **IRepository** interface.

**Loan**

A **Loan** is a **base class** of any **type of loan** and it **should not be able to be instantiated**.

**Data**

* **InterestRate** **– int**
* The amount a bank charges a borrower.
* **Amount - double**
* The amount of the loans offered by the bank.

**Constructor**

A **Loan** should take the following values upon initialization:

**int interestRate, double amount**

**Child Classes**

There are two concrete types of **Loan**:

**StudentLoan**

The student loan has an **interest rate of 1** and an **amount of 10 000**.

**Note:** The Constructor **should take no values** upon initialization.

**MortgageLoan**

The mortgage loan has an **interest rate of 3** and an **amount of 50 000**.

**Note:** The Constructor **should take no values** upon initialization.

**Client**

A **Client** is a **base class** of any **type of client** and it **should not be able to be instantiated**.

**Data**

* **Name – string**
* If the **Name** is **null or whitespace,** throw a new **ArgumentException** with the message:

"Client name cannot be null or empty."

* **Id - string**
* If the ID is **null or whitespace**, throw a new **ArgumentException** with the message:

"Client’s ID cannot be null or empty."

* **Interest – int**
* The client’s interest. ***Be careful with the access modifier***.
* **Income - double**
* The client’s income
* If the income is **below or equal** to **0,** throw an **ArgumentException** with the message:

"Income cannot be below or equal to 0."

**Behavior**

**void IncreaseInterest()**

The **IncreaseInterest()** is an **abstract** method that ***increases the client’s interest***. Keep in mind that the child classes of **Client *will implement the method differently***.

**Constructor**

A **Client** should take the following values upon initialization:

**string name, string id, int interest, double income**

**Child Classes**

There are several concrete types of **Client**:

**Student**

Has an initial **interest of 2 percent.**

**This class will be only accepted in combination with BranchBank. For more clarity, see the AddClient command in the business logic section of this document.**

The constructorshould take the following values upon initialization:

**string name, string id, double income**

**Behavior**

**void IncreaseInterest()**

* The method **increases** the client’s interest by **1 percent**.

**Adult**

Has an initial **interest of 4 percent.**

**This class will be only accepted in combination with CentralBank. For more clarity, see the AddClient command in the business logic section of this document.**

The constructorshould take the following values upon initialization:

stringname,stringid, double income

**Behavior**

**void IncreaseInterest()**

* The method **increases** the client’s interest by **2 percent**.

**Bank**

**Bank** is a **base class** of any **type of bank** and it **should not be able to be instantiated**.

**Data**

* **Name - string**
* If the name **is null or whitespace,** throw an **ArgumentException** with the message:

"Bank name cannot be null or empty."

* All names are unique.
* **Capacity - int**
* The **number** of **clients** а **Bank** **can have.**
* **Loans - IReadOnlyCollection<ILoan>**
* **Clients - IReadOnlyCollection<IClient>**

**Behavior**

**double SumRates()**

**Returns the sum** of the **interest rates** of each **loan in the Bank.**

**void AddClient(IClient client)**

**Adds** a **Client** in the **Bank** if there is a **capacity** for it.

If there is **not enough capacity** to **add** the **Client** to the **Bank, throw an ArgumentException** with **the following message:**

* **"**Not enough capacity for this client.**"**

**void RemoveClient(IClient client)**

Removes a **Client** from the **Bank**. It is important to note that **you will always receive clients** that have **already** been **created within the application**.

**void AddLoan(ILoan loan)**

**Adds** a **Loan** in the **Bank**.

**String GetStatistics()**

**Returns** a **string** with **information** about the **Bank** in the format below.

**"**Name: **{bankName}**, Type: **{bankTypeName}**Clients: **{clientName1}, {clientName2} ... /** Clients: none  
Loans: **{loansCount}**, Sum of Rates: **{sumOfInterestRates}"**

**NOTE: Do not use** "\r\n" **for a new line.**

**Constructor**

A **Bank** should take the following values upon initialization:

stringname,intcapacity

**Child Classes**

There are 2 concrete types of **Bank**:

**BranchBank**

Has **25 capacity.**

The constructorshould take the following values upon initialization:

stringname

**CentralBank**

Has **50 capacity.**

The constructorshould take the following values upon initialization:

stringname

**LoanRepository**

The **LoanRepository** is a **repository** for all the **loans** that are in the **banks**.

**Data**

* **Models** - **IReadOnlyCollection<ILoan>**

**Behavior**

**void AddModel(ILoan loan)**

* **Adds** a **loan** to the **collection**.

**boolean RemoveModel(ILoan loan)**

* **Removes** a **loan** from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**ILoan FirstModel(string name)**

* **Returns** the **first** **loan** of the **given type**, if there is any. **Otherwise**, returns **null**. Within the method implementation, you can **access the name of the loan's type** using **"GetType().Name"**. This allows you to identify the loan type and perform the necessary logic to retrieve the first loan of that type.

**BankRepository**

The **BankRepository** is a **repository** for all the **banks** that are created in the **application**.

**Data**

* **Models** - **IReadOnlyCollection<IBank>**

**Behavior**

**void AddModel(IBank bank)**

* **Adds** a **bank** to the **collection**.

**boolean RemoveModel(IBank bank)**

* **Removes** a **bank** from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IBank FirstModel(string name)**

* **Returns** the **first** **bank** with the **given name**, if there is such bank. **Otherwise**, returns **null**.

**Task 2: Business Logic (150 points)**

**The Controller Class**

The program’s business logic should be concentrated around several **commands**, which you have to implement in the correct class.

**NOTE: Do not use** "\r\n" **for a new line.**

The interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of the **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "this.controller = new Controller();" line.

**Data**

You will need some private fields in your controller class:

* **loans** - **LoanRepository**
* **banks** - **BankRepository**

**Commands**

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**.

**AddBank Command**

**Parameters**

* **bankTypeName** - **string**
* **name** - **string**

**Functionality**

**Creates a bank** from the appropriate type and **adds** itto the **BankRepository**.

If the **bankTypeName** is an **invalid type in the application**, **throw an ArgumentException with the following message**:

* **"**Invalid bank type.**"**

If the **Bank** is **added successfully**, the method should **return** the following **string**:

* **"{bankTypeName}** is successfully added.**"**

**AddLoan Command**

**Parameters**

* **loanTypeName** - **string**

**Functionality**

**Creates** a **loan** from the appropriate type and **adds** it to the **LoanRepository**.

If the **loanTypeName** is an **invalid type in the application**, throw an **ArgumentException** with the following message:

* **"**Invalid loan type.**"**

If the **Loan** is **added successfully**, the method should **return** the following **string**:

* **"{loanTypeName}** is successfully added.**"**

**ReturnLoan Command**

**Parameters**

* **bankName - string**
* **loanTypeName - string**

**Functionality**

**Adds** the appropriate **ILoan**, returned by a client, to the **Bank** with the **given name**. You have to remove the **Loan** from the **LoanRepository** if the insert is **successful**.

It is important to note that **the bank referenced by the bankName** parameter will **always exist** in the **BankRepository**. Therefore, you can assume that **the specified bank is valid and present**.

If there is **no such loan**, you have to **throw an ArgumentException** with **the following message**:

* **"**Loan of type **{loanTypeName}** is missing.**"**

If **no exceptions** are **thrown, return** the **String**:

* **"{loanTypeName}** successfully added to **{bankName}**.**"**

**AddClient Command**

**Parameters**

* **bankName** - **string**
* **clientTypeName** - **string**
* **clientName** - **string**
* **id** - **string**
* **income** - **double**

**Functionality**

* Ifthe given **clientTypeName is not recognized** as a **valid type** in the application, the method should **throw an ArgumentException** with **the following message**:

**"**Invalid client type.**"**

* **Select** from the **BankRepository** the **bank with the given bankName**.
* **I**f the given **clientTypeName** is **NOT** a **valid client type** for the **selected bank**, the following message is returned:

**"**Unsuitable bank.**"**

* Otherwise **creates and adds** client from the **appropriate type** to the **Bank** with the **given name**. The following message should be returned:

**"{clientTypeName}** successfully added to **{bankName}**.**"**

**FinalCalculation Command**

**Parameters**

* **bankName** - **string**

**Functionality**

Calculates all funds that have passed through the **Bank** with the given name. It is calculated from the sum of all **income from clients** and **amount from loans** in the **Bank**.

**Return** a **string** in the following **format**:

* **"**The funds of bank **{bankName}** are **{funds}."**
* The **funds** should be **formatted** to the **2nd decimal place**!

**Statistics Command**

**Functionality**

Returns information about each bank. You can use Bank's **GetStatistics()** method to implement the current functionality.

**"**Name: **{bankName}**, Type: **{bankType}**Clients: **{clientName1}**, **{clientName2} ... /** Clients: none  
Loans: **{loansCount}**, Sum of Rates: **{sumOfInterestRates}**

Name: **{bankName}**, Type: **{bankType}**Clients: **{clientName1}**, **{clientName2} ... /** Clients: noneLoans: **{loansCount}**, Sum of Rates: **{sumOfInterestRates}**

**..."**

**NOTE: Do not use** "\r\n" **for a new line.**

**End Command**

Ends the program.

**Input / Output**

You are provided with one interface, which will help you with the correct execution process of your program. The interface is **Engine** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

**Input**

Below, you can see the **format** in which **each command** will be given in the input:

* **AddBank** **{bankTypeName} {name}**
* **AddLoan** **{loanTypeName}**
* **ReturnLoan** **{bankName} {loanTypeName}**
* **AddClient {bankName} {clientTypeName} {clientName} {id} {income}**
* **FinalCalculation {bankName}**
* **Statistics**
* **End**

**Output**

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

**Examples**

|  |
| --- |
| **Input** |
| **AddBank BranchBank DSKBank**  **AddBank CentralBank Unicredit**  **AddBank CentralBank Fibank**  **AddLoan StudentLoan**  **AddLoan MortgageLoan**  **AddLoan MortgageLoan**  **ReturnLoan DSKBank StudentLoan**  **ReturnLoan Unicredit StudentLoan**  **ReturnLoan DSKBank MortgageLoan**  **ReturnLoan Fibank MortgageLoan**  **AddClient DSKBank Student Sarah 10A2AFBBAG 5421.5**  **AddClient DSKBank Student Tom 54AABAG75 2341.1**  **AddClient Fibank Adult Peter 6GSFAAZZ12 125054**  **FinalCalculation DSKBank**  **Statistics**  **End** |
| **Output** |
| **BranchBank is successfully added.**  **CentralBank is successfully added.**  **CentralBank is successfully added.**  **StudentLoan is successfully added.**  **MortgageLoan is successfully added.**  **MortgageLoan is successfully added.**  **StudentLoan successfully added to DSKBank.**  **Loan of type StudentLoan is missing.**  **MortgageLoan successfully added to DSKBank.**  **MortgageLoan successfully added to Fibank.**  **Student successfully added to DSKBank.**  **Student successfully added to DSKBank.**  **Adult successfully added to Fibank.**  **The funds of bank DSKBank are 67762.60.**  **Name: DSKBank, Type: BranchBank**  **Clients: Sarah, Tom**  **Loans: 2, Sum of Rates: 4**  **Name: Unicredit, Type: CentralBank**  **Clients: none**  **Loans: 0, Sum of Rates: 0**  **Name: Fibank, Type: CentralBank**  **Clients: Peter**  **Loans: 1, Sum of Rates: 3** |

|  |
| --- |
| **Input** |
| **AddBank BranchBank DSKBank**  **AddBank CentralBank Fibank**  **AddLoan StudentLoan**  **AddLoan MortgageLoan**  **AddLoan MortgageLoan**  **ReturnLoan DSKBank StudentLoan**  **ReturnLoan Fibank StudentLoan**  **ReturnLoan Fibank MortgageLoan**  **AddClient Fibank Student Maria 54TAF433 346.7**  **AddClient Fibank Adult Peter 65GTTHA134 5643.1**  **FinalCalculation Fibank**  **Statistics**  **End** |
| **Output** |
| **BranchBank is successfully added.**  **CentralBank is successfully added.**  **StudentLoan is successfully added.**  **MortgageLoan is successfully added.**  **MortgageLoan is successfully added.**  **StudentLoan successfully added to DSKBank.**  **Loan of type StudentLoan is missing.**  **MortgageLoan successfully added to Fibank.**  **Unsuitable bank.**  **Adult successfully added to Fibank.**  **The funds of bank Fibank are 55643.10.**  **Name: DSKBank, Type: BranchBank**  **Clients: none**  **Loans: 1, Sum of Rates: 1**  **Name: Fibank, Type: CentralBank**  **Clients: Peter**  **Loans: 1, Sum of Rates: 3** |

**Task 3: Unit Tests (100 points)**

You will receive a skeleton with one class inside it. **CoffeeMat** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended.

* **Do NOT CHANGE OR REMOVE ANY namespaces or usings.**
* **NOTE: Do not use** "\r\n" **for a new line.**
* In Judge, you upload **.zip** **(**with **VendingRetail.Tests** inside**)** from the **skeleton**