# Java OOP Exam – 5 August 2023



## 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 students and adults, 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 bank** package 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.

## Task 1: Structure (50 points)

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

It is not required to implement your structure with Engine, ConsoleReader, ConsoleWriter, and enc. It's good practice but it's not required.

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

There should also be **LoanRepository**.

### BaseLoan

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

#### Data

* **interestRate** **- int**
* **amount – double**
  + The amount of the loans offered by the bank.

#### Constructor

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

(int interestRate, double amount)

#### Child Classes

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

##### 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.

### BaseClient

BaseClient 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 **IllegalArgumentException** with a message:

"**Client name cannot be null or empty.**"

* **ID - String**
  + If the ID **is null or whitespace,** throw a **IllegalArgumentException** with a message:

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

* **interest – int**
  + The client’s interest.
* **income - double**
  + The client’s income.
  + If the income is below or equal to **0,** throw an **IllegalArgumentException** with the message:

"**Client income cannot be below or equal to 0.**"

#### Behavior

##### void increase()

The **increase()** method increases the **Client’s** interest. Keep in mind that some **Client** can implement the method differently.

#### Constructor

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

(String name, String ID, int interest, double income)

#### Child Classes

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

##### Student

Has **initial interests of 2 percent.**

**Can only live in BranchBank!**

The constructorshould take the following values upon initialization:

**(String name, String ID, double income)**

#### Behavior

**void increase()**

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

##### Adult

Has **initial interest of 4 percent.**

**Can only live in CentralBank!**

The constructorshould take the following values upon initialization:

**(String name, String ID, double income)**

#### Behavior

**void increase()**

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

### BaseBank

BaseBank 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 **IllegalArgumentException** with a message:

"**Bank name cannot be null or empty.**"

* **capacity - int**
  + The **number** of **clients** а **Bank** **can have.**
* **loans - Collection<Loan>**
* **clients - Collection<Client>**

#### Behavior

##### int sumOfInterestRates()

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

##### void addClient(Client 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 IllegalStateException** with **the following message:**

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

##### void removeClient(Client 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(Loan loan)

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

##### String getStatistics()

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

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

**Note:** I remind you that there are **two bank types – CentralBank** and **BranchBank.**

#### Constructor

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

**(String name, int capacity)**

#### Child Classes

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

##### BranchBank

Has **25 capacity.**

The constructorshould take the following values upon initialization:

**(String name)**

##### CentralBank

Has **50 capacity.**

The constructorshould take the following values upon initialization:

**(String name)**

### LoanRepository

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

#### Data

* loans - **Collection<Loan>**

#### Behavior

**void addLoan(Loan loan)**

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

**boolean removeLoan(Loan loan)**

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

**Loan findFirst(String type)**

* **Returns** the **first** **loan** of the **given type**, if there is any. **Otherwise**, returns **null**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces that you must implement in the correct classes.

**Note: The** ControllerImpl **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The first interface is Controller. You must create a ControllerImplclass, which implements the interface and implements all its methods. The given methods should have the following logic:

### Data

You need some private fields in your controller class:

* **loans** - **LoanRepository**
* **banks** - **Collection<Bank>**

### Commands

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

#### AddBank Command

##### Parameters

* type - String
* name - String

##### Functionality

**Creates and adds** a Bank to the banks’ collection. **Valid** types are: "**CentralBank**" and "**BranchBank**".

If the **Bank** **type** is **invalid**, you have to **throw a IllegalArgumentException** with **the following message:**

* **"Invalid bank type."**

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

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

#### AddLoan Command

##### Parameters

* **type** - **String**

##### Functionality

**Creates** a **loan** of the **given type** and **adds** it to the **LoanRepository**. **Valid** types are: "**StudentLoan**" and "**MortgageLoan**". If the loan **type** is **invalid**, throw an **IllegalArgumentException** with a message:

* **"Invalid loan type."**

The **method** should **return** the following **string** if the **operation** is **successful**:

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

#### ReturnedLoan Command

##### Parameters

* bankName - String
* loanType - String

##### Functionality

**Adds** the returned Loan 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 IllegalArgumentException** with **the following message**:

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

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

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

#### AddClient Command

##### Parameters

* bankName - String
* clientType - String
* clientName - String
* ID - String
* income - double

##### Functionality

**Creates and adds** the desired Client to the Bank with the **given name**.

**Valid** Client types are: "**Student**" and "**Adult**".

**Note:** The method must first check whether the **client** type is **valid**.

If the **Client** **type** is **invalid**, you have to **throw an IllegalArgumentException** with **the following message**:

* **"Invalid client type."**

If the clientTypeName is not **"Adult" for the CentralBank** or **"Student" for the BranchBank**, the client type is considered unsuitable for the Bank.

If **no errors** are **thrown**, **return** one of the following strings:

* **"Unsuitable bank."** - if the given **Client** **cannot be a user** of the **Bank.**

**For reference:** check their description from **Task 1**.

* **"{clientType} successfully added to {bankName}."** - if the **Client** is **added successfully** in the **Bank.**

#### FinalCalculation Command

##### Parameters

* **bankName** - **String**

##### Functionality

Calculates all funds (**income** from **Client** and **amount** from **Loan**) that have passed through the **Bank** with the given name. It is calculated from the sum of all **Client** and **Loan** prices in the **Bank**.

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

* **"The funds of bank {bankName} is {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 interest rates: {sumOfInterestRates}**

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

**..."**

#### 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** **{type} {name}**
* **AddLoan** **{type}**
* **ReturnedLoan** **{bankName} {loanType}**
* **AddClient {bankName} {clientType} {clientName} {clientID} {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**  **ReturnedLoan DSKBank StudentLoan**  **ReturnedLoan Unicredit StudentLoan**  **ReturnedLoan DSKBank MortgageLoan**  **ReturnedLoan 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 is 67762,60.**  **Name: DSKBank, Type: BranchBank**  **Clients: Sarah, Tom**  **Loans: 2, Sum of interest rates: 4**  **Name: Unicredit, Type: CentralBank**  **Clients: none**  **Loans: 0, Sum of interest rates: 0**  **Name: Fibank, Type: CentralBank**  **Clients: Peter**  **Loans: 1, Sum of interest rates: 3** |

|  |
| --- |
| **Input** |
| **AddBank BranchBank DSKBank**  **AddBank CentralBank Fibank**  **AddLoan StudentLoan**  **AddLoan MortgageLoan**  **AddLoan MortgageLoan**  **ReturnedLoan DSKBank StudentLoan**  **ReturnedLoan Fibank StudentLoan**  **ReturnedLoan 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 is 55643,10.**  **Name: DSKBank, Type: BranchBank**  **Clients: none**  **Loans: 1, Sum of interest rates: 1**  **Name: Fibank, Type: CentralBank**  **Clients: Peter**  **Loans: 1, Sum of interest rates: 3** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **Client** and **Bank**. **Bank** 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. In Judge you upload **.zip** to **bank (**with **BankTests** inside**)** from the **skeleton**.