Engineering Informatic
17 of May, 2024
Submission Deadline: 02/06/2024

BANCO BRIGANTINO

A new bank branch has opened in Bragança, requiring a simple application to manage client transaction balances urgently. This application must enable the addition of new clients and the creation of accounts associated with these clients, which can be of two types: Current Account or Savings.

- Current Account: Each new client should automatically receive a Current Account with an initial balance and a specified credit limit (the maximum credit that can be used; for example, if the value is 900, the account can be operated until the balance reaches -900€).
- Savings Account: Unlike the Current Account, this does not have a credit limit but includes an adjustable interest rate with the client, serving only as additional information about the account.

Both accounts and clients should have an auto-incrementing ID upon creation. Other relevant information to be stored about the client includes their name and phone contact.

Each account must also generate a unique NIB (Bank Identification Number) for each account. The function to generate and validate the NIB is provided at the end of this document and should be implemented in the appropriate classes.

The possible operations on the accounts are only two:

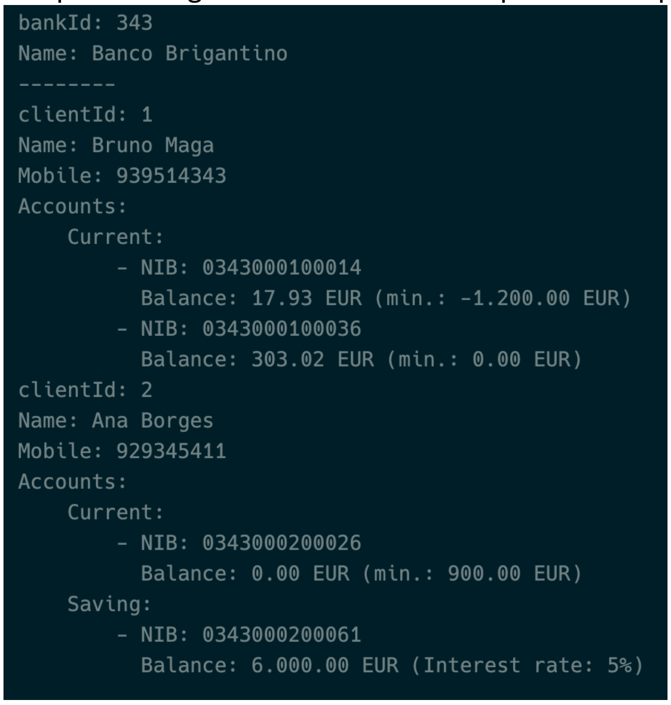
- Deposits
- Withdrawals (available only for Current Accounts)

You need to implement all the necessary methods to create, modify, and display each of the mentioned elements. Error control mechanisms should be implemented whenever possible, such as:

The withdrawal operation should not proceed if the amount exceeds the account's credit limit.

Attempting to deposit into a non-existent account or with an invalid NIB should be prevented.

An example output after performing a series of described operations is presented below.



```
bankId: 343
Name: Banco Brigantino
--------
clientId: 1
Name: Bruno Maga
Mobile: 939514343
Accounts:
    Current:
        - NIB: 0343000100014
          Balance: 17.93 EUR (min.: -1.200.00 EUR)
        - NIB: 0343000100036
          Balance: 303.02 EUR (min.: 0.00 EUR)
clientId: 2
Name: Ana Borges
Mobile: 929345411
Accounts:
    Current:
        - NIB: 0343000200026
          Balance: 0.00 EUR (min.: 900.00 EUR)
    Saving:
        - NIB: 0343000200061
          Balance: 6.000.00 EUR (Interest rate: 5%)
```

Tasks to be developed

1. Start by modeling the problem, drawing the UML class diagram, and considering only the following classes: Bank, Customer, Account, Order, and Time.

   (Astah[1], Visual Paradigm[2] or any other UML modeling tool that allows you to save the result in jpg, png or pdf format can be used to prepare the class diagram.)

2. Implement in C++ a solution to the problem that is a faithful translation of the class diagram created.

Considerations to consider when implementing

1. An application for the console should be developed in *MS Visual Studio,* version 2015 or later.
2. Include in the solution a small *main* that makes use of all the features of the application.
3. Both the project and the solution, created in *Visual Studio,* must be called trabPOO, and be properly anonymized. If they contain any element that allows the identification of the authors (including in the class diagram), 1 value will be deducted from the classification of the work.
4. Each class must be defined using two files, one with the declaration of the class (*.h) and the other with its implementation (*.cpp). Both files must have the exact same class name.
5. In the implementation of the collections, they must use the template Colecao.h (available at ipb.virtual) and are totally forbidden to change anything in that template.
6. In order to simplify the implementation, you should use the string class for the date type.

General considerations

1. The work must be carried out by groups of 2 students.
2. Only works whose implementation does not present any compilation or *linking* error and with a minimum of perfectly operational functionalities will be accepted for evaluation.
3. It is expressly forbidden to copy all or part of code from sources other than the documentation provided by the teachers of the curricular unit.
4. Students may have to defend their work, in person or by videoconference, on a date to be set by the teacher, in order to demonstrate the ability to implement the code, understand it and explain it.
5. For questions and additional clarifications about the work, you should use the discussion forum created on the http://virtual.ipb.pt/ platform: OOP (23/2.4) > Forums > Practical Work.

   (Given the large number of students enrolled, and so that everyone benefits from the clarifications that will be provided, no doubts about the work will be answered by email.)

Submission Rules

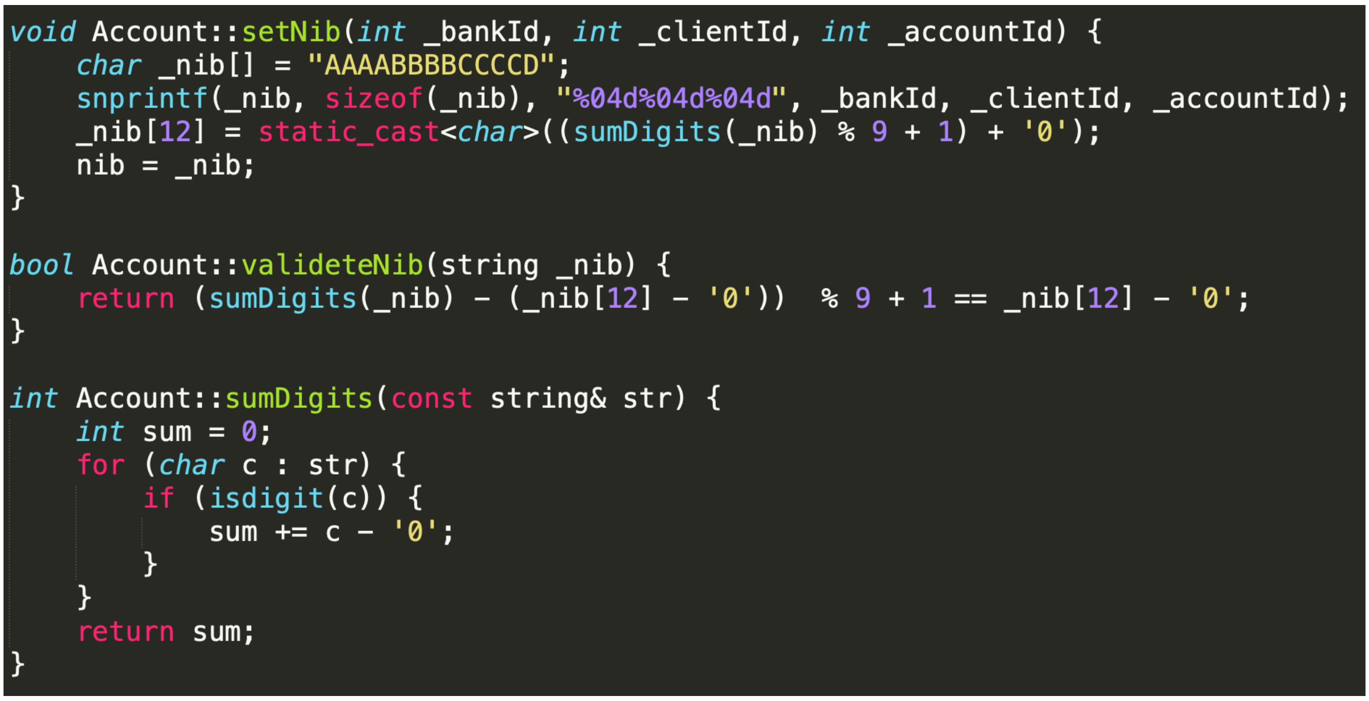
1. The work must be submitted only by one of the members of the working group, on the e-learning portal (http://virtual.ipb.pt/, option Activities), within the established deadline. Papers sent by email or other forms of submission will not be accepted.
2. Two files must be submitted:

[1] https://astah.net/products/free-student-license/
[2] https://www.visual-paradigm.com/solution/freeumltool/

- trabPOO.zip – Archived folder of the Visual C++ solution, after excluding, if any, the files with the extension ".sdf", ". VC" and ". VC.db" (and any other auto-create file that takes up a lot of space) and the subfolders named "Debug", "Release", "ipch", "x64" and ".vs" (note that this last subfolder may be hidden);

  It is also inside this folder (trabPOO.zip) that you should place the file with the class diagram, which should be named ClassDiagram (with extension .jpg, .png or .pdf).
- autores.txt – text file containing only the name and mechanographic number of the two authors of the work.

3. The work can only be submitted with a maximum delay of 3 days, in which case the subtraction of one value for each day of delay.



```cpp
void Account::setNib(int _bankId, int _clientId, int _accountId) {
    char _nib[] = "AAAABBBBCCCCD";
    snprintf(_nib, sizeof(_nib), "%04d%04d%04d", _bankId, _clientId, _accountId);
    _nib[12] = static_cast<char>((sumDigits(_nib) % 9 + 1) + '0');
    nib = _nib;
}

bool Account::valideteNib(string _nib) {
    return (sumDigits(_nib) - (_nib[12] - '0'))  % 9 + 1 == _nib[12] - '0';
}

int Account::sumDigits(const string& str) {
    int sum = 0;
    for (char c : str) {
        if (isdigit(c)) {
            sum += c - '0';
        }
    }
    return sum;
}
```