Université Grenoble Alpes 2018-2019
M1 MSIAM

# Algorithms and software tools - C++
# Lab 4

`

*The purpose of this Lab is to define some classes and functions to help a banker manage the accounts of his clients.*

**Exercise 1.** *Inheritance*

Every ***banking account*** will be characterized by an identifier which is a *number* (an unsigned integer) and it has a current *balance* (a real number). For every account, it will be possible to perform the following operations: credit an amount of money, debit an amount of money, get the identifier, and display the attributes of the account.
For every kind of account, *crediting* an amount of money simply corresponds to adding this amount to the current balance.
Depending on the type of account, *debiting* an amount of money has a specific definition (see below). In the most general class, we will consider in this Lab session that it simply corresponds to removing this amount from the current balance.

There are two main types of banking accounts: current accounts and savings accounts.

- for ***current accounts***, debiting an amount of money *a* is defined as follows: if the balance is greater than *a* (i.e., there is enough money) then the debit is performed as expected, if there is not enough money but the balance is positive then the account is debited from the remaining balance, and finally if the balance is zero then no debit is possible. The corresponding function will return the amount of money that has actually been debited.
- ***savings accounts*** have an additional attribute which is the *interest rate*. There are two types of savings accounts, which have specific interest rates: ***blocked accounts*** and ***unblocked accounts***. For blocked accounts, the interest rate IRATEBLOCKED is 4%, and for unblocked accounts, the interest rate IRATEUNBLOCKED is 2%. For every savings account, it will be possible to *add interests* to the current balance.
  It is impossible to debit blocked accounts (a specific code -1 will be used as returned value). Debiting unblocked accounts of an amount of money *a* is defined as follows: if the balance is greater than *a* (i.e., there is enough money) then the debit is performed as expected, otherwise no debit is possible. The function still returns the amount of money that has actually been debited.

**1.** What will be the class hierarchy to describe those different types of accounts?
Knowing that we want to give the possibility to directly access the value of the balance, from any method of any of these classes, which data/methods will be private, protected, public in the most general class?

**2.** Knowing also that we want to give the possibility to get the identifier (i.e. number) of any kind of account *acc* using the syntax *acc()*, write a file `account.h` that contains:
- the definition of the two macros IRATEBLOCKED with value 4, and IRATEUNBLOCKED with value 2
- the declarations of the classes specified above. In particular we must have, among the methods of the most general class Account:

```
void credit(double a);     // credits the amount of money a
double debit(double a);    // debits the amount of money a
void print(ostream &o);    // displays the attributes of the account on ostream o
```

**3.** Write the file `account.cpp` that contains the definitions of the methods of those classes.

**Exercise 2.** *Aggregation, overloading*

Now we consider a characterization of the ***clients***. In this Lab session, every client will be characterized by his *name* (a string), his *identifier* (an unsigned integer), a *pointer to a current*

*account*, and a *pointer to an unblocked savings account* (those pointers will be null if the client does not have the corresponding type of account).
It will be possible to: get his identifier, credit his *current account* of an amount of money *a*, credit his *savings account* of an amount of money *a*, try to debit his *current account* of an amount of money *a*, try to debit his *savings account* of an amount of money *a*, and display his attributes (including the attributes of his accounts) by means of the stream insertion operator <<

**1.** Write a file `client.h` that contains the declaration of this class Client.

**2.** Write the file `client.cpp` that contains the definitions of the methods of this class.

**Exercise 3.** *Collection*

A ***bank*** is characterized by a *collection of clients*. An attribute of type "map" will be used to model this collection. A *map* is an associative container of the STL that enables to associate data with keys (http://www.cplusplus.com/reference/map/map/), see a brief exemplified description in the Appendix. Elements are automatically sorted according to the keys.
Here the *data* stored in the map will be *pointers to Clients*, and the *key* associated with each client will be his *identifier*. A method *InsertClient* will give the possibility to insert an existing client into this map.
Here too, we will give the possibility to display the attributes of instances of *bank* by means of the stream insertion operator <<

**1.** Give the declaration of this class Bank, and the definitions of the methods and functions.

**2.** Write a *main* function that creates and modifies instances of bank accounts, creates instances of Client, insert them in an instance of Bank, modifies, and prints them.

**Appendix.** template < class Key, class T > class map;

Maps are associative containers that store elements formed by a combination of a **key value** (of type Key) and a **mapped value** (of type T), following a specific order. In a map, the key values are generally used to sort and uniquely identify the elements, while the mapped values store the content associated to this key.
When you dereference an iterator over the map class, you get a "pair", which essentially has two members, *first* and *second*. *First* corresponds to the key, *second* to the value.
There are several ways to access the elements of a map. You can simply treat the map like a normal array and use brackets.

Example (http://www.yolinux.com/TUTORIALS/CppStlMultiMap.html) :

```cpp
#include <string.h>
#include <iostream>
#include <map>
#include <utility>

using namespace std;

int main()
{
   map<int, string> Employees;

   // 1) Assignment using array index notation
   Employees[5234] = "Mike C.";
   Employees[3374] = "Charlie M.";
   Employees[1923] = "David D.";
   Employees[7582] = "John A.";
   Employees[5328] = "Peter Q.";

   cout << "Employees[3374]=" << Employees[3374] << endl << endl;

   cout << "Map size: " << Employees.size() << endl;

   for( map<int,string>::iterator ii=Employees.begin(); ii!=Employees.end(); ++ii)
   {
       cout << (*ii).first << ": " << (*ii).second << endl;
   }
}
```