Course project

CSC 2110

# Information

**Due date:** Monday, Dec 6, 2021

**Points:** 100

**Learning objectives:**

1. Understand user defined classes and be able to apply them in C++ programs.
2. Understand inheritance and composition and be able to apply them in C++ classes.
3. Understand pointer data type and become aware of the features of classes with

pointers member variables.

1. Understand polymorphism conceptually and be able to create C++ classes using

polymorphism.

1. Understand operators overloading and be able to apply them in C++ programs.
2. Incorporate exception handling in object-oriented programs.
3. Understand various Sorting and searching algorithms and be able to apply them in

C++ programs.

1. Use data structures (linked list) and standard template library (vector type).
2. Apply problem solving strategies to analyze, design, implement, document, and test

a software application.

# Instructions

## Project

Create a Visual Studio C++ project. Name it using the following format:

firstname\_lastname\_project

(e.g. my name is Alberto Cahdez, then the project name is alberto\_cahdez\_project).

## Bank account system

The objective of the project is to implement a micro bank account system which manipulates the accounts of one client. In this system, the user creates a client, adds and removes bank accounts, and execute other operations with those accounts.

## Account abstract class

|  |
| --- |
| Account |
| #balance: float |
| +getBalance() const: float  +deposit(float *amount*): void |

Account class is an abstract class i.e., all functions are pure virtual functions.

Account has a protected float variable called balance. Its accessor is getBalance.

The deposit function increases the balance by the value of *amount*.

### Account’s business rules

balance cannot be less than zero.

## Checking account class

|  |
| --- |
| CheckingAccount |
| +withdraw(float *amount*): void  +CheckingAccount(float *amount*) |

CheckingAccount class is a derived class from Account class. Use public as the access specifier for the inheritance.

The withdraw function decreases the balance by the value of *amount*.

The constructor initializes the balance in the account using *amount*.

### Checking account’s business rules

In the withdraw function, if *amount* is greater than the balance, then the balance is not affected, and it raises an exception with the string message “It cannot withdraw an amount greater than the balance. No changes were done to the balance.”

## SavingAccount class

|  |
| --- |
| SavingAccount |
| -SavingRate: float |
| +setSavingRate(float *rate*): void  +getSavingRate() const: float  +transfer(CheckingAccount& checkingAcc,float *amount*): void  +SavingAccount(float *amount*,float *rate*) |

SavingAccount class is a derived class from Account class. Use public as the access specifier for the inheritance.

The constructor initializes balance, and SavingRate with *amount* and *rate*, respectively.

Saving Accounts has a private float member called SavingRate which represents the saving account rate. Its accessor and mutator are getSavingRate and setSavingRate, respectively.

The transfer function decreases the balance by the value of *amount*. In addition, this function deposits *amount*’s value into *checkingAcc*.

### Saving account’s business rules

In the transfer function, if *amount* is greater than the balance, then the balance is not affected, and it raises an exception with the string message “It cannot withdraw an amount greater than the balance. No changes were done to the balance.”

## Client class

|  |
| --- |
| Client |
| -id: int  -firstName: string  -lastName: string  +checking: CheckingAccount\*  +saving: SavingAccount\* |
| +setID(int *i*): void  +getID() const: int  +setName(string *fn*,string *ln*): void  +getFirstName() const: string  +getLastName() const: string  +Client(string *i*, string *fn*, string *ln*) |

The Client class contains three private variable members named id, firstName, and lastName. Also, it contains 2 pointers to CheckingAccount and SavingAccount objects.

setID and getID functions are id’s mutator and accessor, respectively.

setName function is a mutator for both firstName and lastName. *fn* and *ln* correspond to these variables respectively. getFirstName and getLastName are the corresponding accessors.

The constructor initializes id, firstName, and lastName using i, fn, and ln respectively; and set checking and saving to null values for pointers.

## BankSystem class

|  |
| --- |
| BankSystem |
| -log: vector<string>  -oneClient: Client |
| +start(): void  -AddCheckingDialog(): void  -RemoveCheckingDialog(): void  -DepositCheckingDialog(): void  -WithdrawCheckingDialog(): void  -AddSavingDialog(): void  -RemoveSavingDialog(): void  -DepositSavingDialog(): void  -TransferDialog(): void  +BankSystem(string *i*, string *fn*, string *ln*) |

### Constructor

The constructor initializes the Client object with the input parameters.

### start

The start function will display the following menu in a loop until the user exits. When the user exits the menu, display the content of the log vector which contains all the actions that occurred in the program.

|  |
| --- |
| = Menu =  1. Add a Checking account  2. Remove Checking account  3. Deposit into Checking Account  4. Withdraw from Checking Account  5. Add a Saving account  6. Remove Saving account  7. Deposit into Saving account  8. Transfer from Saving to Checking account  9. Exit  Choose an option [1-9]: |

If the user chooses 1, call AddCheckingDialog; 2, call RemoveCheckingDialog; 3, call DepositCheckingDialog; 4, call WithdrawCheckingDialog; 5, call AddSavingDialog; 6, call RemoveSavingDialog; 7, call DepositSavingDialog; 8, call TransferDialog; 9, exit the menu loop.

### AddCheckingDialog

The AddCheckingDialog function will check the oneClient’s checking member. If it is not null, display that “The client already has a checking account”, log\* “Invalid attempt to overwrite a checking account.” and return to the menu.

If it is null, ask for the initial balance; then create dynamically the Checking account object and set it in the checking member of the client. Also, log\* “Checking account successfully created.”.

### RemoveCheckingDialog

The RemoveCheckingDialog function will check the oneClient’s checking member. If it is null, display that “The client does not have a checking account”, log\* “Invalid attempt to remove a non-existing checking account.” and return to the menu.

If it is not null, remove the checking account from memory. Also, log\* “Checking account successfully removed.”.

### DepositCheckingDialog

The DepositCheckingDialog function will check the oneClient’s checking member. If it is null, display that “The client does not have a checking account”, log\* “Invalid attempt to deposit in a non-existing checking account.” and return to the menu.

If it is not null, ask for the amount to deposit, and add it to the checking account. Also, log\* “$[amount] successfully deposited in checking account.” where [amount] is the value asked to the user.

### WithdrawCheckingDialog

The WithdrawCheckingDialog function will check the oneClient’s checking member. If it is null, display that “The client does not have a checking account”, log\* “Invalid attempt to withdraw from a non-existing checking account.” and return to the menu.

If it is not null, ask for the amount to withdraw, and add it to the checking account. Also, log\* “$[amount] successfully withdrawn from checking account.” where [amount] is the value asked to the user.

### AddSavingDialog

The AddSavingDialog function will check the oneClient’s saving member. If it is not null, display that “The client already has a saving account”, log\* “Invalid attempt to overwrite a saving account.” and return to the menu.

If it is null, ask for the initial balance and saving rate; then create dynamically the Saving account object and set it in the saving member of the client. Also, log\* “Saving account successfully created.”.

### RemoveSavingDialog

The RemoveSavingDialog function will check the oneClient’s saving member. If it is null, display that “The client does not have a saving account”, log\* “Invalid attempt to remove a non-existing saving account.” and return to the menu.

If it is not null, remove the saving account from memory. Also, log\* “Saving account successfully removed.”.

### DepositSavingDialog

The DepositSavingDialog function will check the oneClient’s saving member. If it is null, display that “The client does not have a saving account”, log\* “Invalid attempt to deposit in a non-existing saving account.” and return to the menu.

If it is not null, ask for the amount to deposit, and add it to the checking account. Also, log\* “$[amount] successfully deposited in saving account.” where [amount] is the value asked to the user.

### TransferDialog

The TransferDialog function will check the oneClient’s saving member. If it is null, display that “The client does not have a saving account”, log\* “Invalid attempt to use a non-existing saving account.” and return to the menu.

Then, this function will check the Client’s checking member. If it is null, display that “The client does not have a checking account”, log\* “Invalid attempt to use a non-existing checking account.” and return to the menu.

If saving and checking are not null, ask for the amount to transfer, and withdraw it from the saving account and add it to the checking account. Also, log\* “$[amount] successfully transferred from saving to checking account.” where [amount] is the value asked to the user.

\* Log means to add the string to the log vector.

### Class files

Classes must be separated in a .h file (header), and a .cpp (source code).

# The main program

In the main program, create a BankSystem object and provide an arbitrary id, your first name, and your last name as input parameters. Then, execute start.

# Submission

Include the appropriate files in the VS project to compile and execute it.

Submit your project in a ZIP folder.