

Project Report

Deadline of the exercise : 31st of January 2022.

Name and Surname :

Kamel Bendimerad.

Project's topic : Bank management system with a trading and a savings account.

Task Topic :

The idea behind this project was to create a bank management system that will give the user the ability to log in and register for three different account types, using a graphical user interface made from the Qt open source library .

Project analysis :

Data structures used in the project :

- In the bank class there was a preference of choosing a map, the keys being the account username and referring to account objects which are pointers since it is a better approach towards accessing child classes, and according to my research, probably the only way possible.

- Stocks are in the form of a data structure in order to unite all the available stocks with the same set of variables. (Share, stock symbol..)
- All of the account informations are stored in a file, instead of a database, in order to integrate the use of the streams which was a new concept for students from this semester.

Algorithms which are used to solve the problem :

For this project we only used an algorithm to log in which had some level of complexity to come up with since it needed the polymorphism integration.

Log in Algorithm :

Scans through the Bank's file and checks the account type of this user and therefore open the exact window in order to access the account's functionalities in its window.

```

MainWindow *w = nullptr;

if (username.size() && password.size() != 0) { // If username & password are not empty not empty

    NormalAccount *account = bank->getAccount(username); // Get the account class from the bank account map

    // Check the account type and open appropriate window

    // UPDATED
    // Giving the parent reference so that whenever MainWindow is deleted it's child will also be deleted. (A Qt parent child fun
    if (account->getAccountType() == Normal) {
        w = new NormalWindow(account, bank, this);
    } else if (account->getAccountType() == Investment) {
        w = new InvestmentWindow(dynamic_cast<InvestmentAccount*>(account), bank, this);
    } else if (account->getAccountType() == Savings) {
        w = new SavingWindow(dynamic_cast<SavingAccount*>(account), bank, this);
    }
}

if (w) {
    w->show(); // Show the window if it is not nullptr
}
}

```

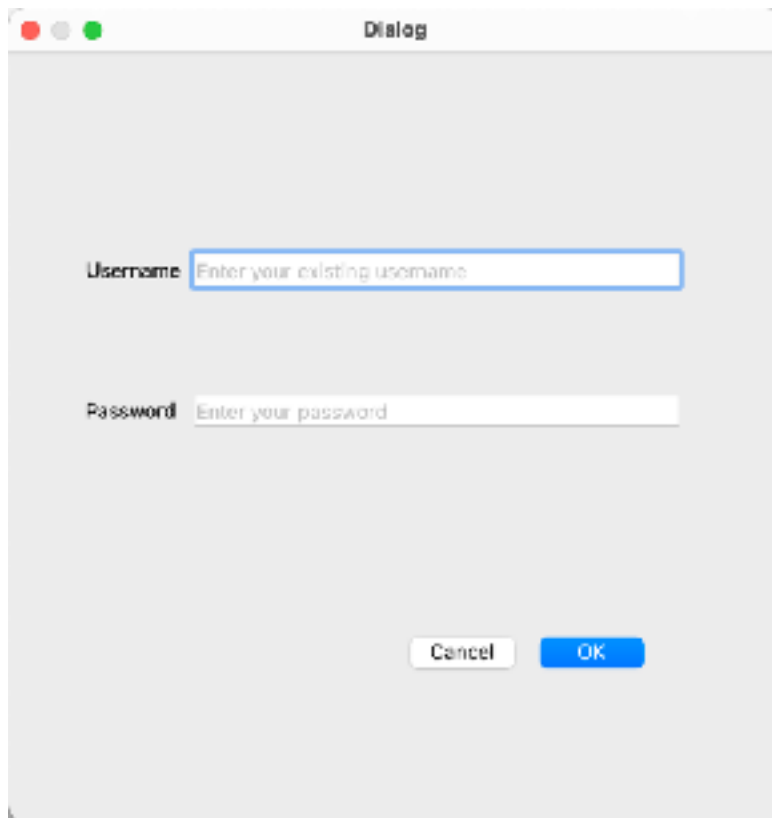
External specification : (user's manual) :

With this program, each user can create an account in this Bank by registering and providing the necessary data, and specifying the account type they wish to create. (accessible after clicking on the register and create an account button).

The image displays a software interface for a bank management system. It consists of two overlapping windows:

- MainWindow:** The background window with a title bar containing three colored buttons (red, yellow, green). The main content area has a light gray background and displays the text "Welcome to to the Bank Managemnt System !". Below this text are two white buttons with black text: "Register and Create a sew accoust" and "Log in to your existing account".
- Dialog:** A foreground window with a title bar containing three colored buttons (red, gray, green). The main content area has a light gray background and contains the following fields and controls:
 - Enter Username :** A text input field with the placeholder text "Enter your username".
 - Enter Password :** A text input field with the placeholder text "Provide a strong password".
 - Initial balance :** A text input field with the placeholder text "enter your initial balance".
 - Select account type :** A section containing three radio buttons:
 - ☒ Normal Account
 - ☐ Investment Account
 - ☐ Salary and Saving Account
 - Normal account is selected by default**: A text label below the radio buttons.
 - Buttons:** At the bottom right, there are two buttons: "Cancel" (white with black text) and "OK" (blue with white text).

After registering, each account has to login to their specific windows, so they need to sign up using the login dialog, with the same password they used while registering.



The image shows a standard macOS-style dialog box titled "Dialog". It has a light gray background and a title bar with three colored window control buttons (red, yellow, green) on the top left. The dialog contains two text input fields. The first field is labeled "Username" and has a placeholder text "Enter your existing username". The second field is labeled "Password" and has a placeholder text "Enter your password". At the bottom right of the dialog, there are two buttons: a "Cancel" button with a light gray background and a blue "OK" button.

After that, for the Normal account, only simple functionalities are present in their interfaces, each normal account window will present to the user their username, balance, the transfer option and the ability to log out from the window.

For the savings and salary account, each user will have the ability to control his financial situation, after they earn their salary, taking a margin from their balance to the savings balance and take away their monthly expenses, they can know which is the final balance that they can currently use :

The screenshot displays a 'Saving Window' application interface. It features a light gray background with various text labels and input fields. At the top left, there are three colored window control buttons (red, yellow, green). The title 'Saving Window' is centered at the top. The interface is organized into several sections: a top section with 'Username : pawel' and 'Expenses : 399'; a middle section with 'Balance : 900677' and 'Monthly Salary : 200' (in a text box); a right section with 'SAVING BALANCE : 9962.01' and 'FINAL USABLE MONEY : 29155.7'; a bottom-left section with 'Recipient's username : ahmad' and a 'Transfer' button; a bottom-center section with 'Amount : 20000' (in a text box) and a 'Log out' button; and a bottom-right section with 'monthly input to our savings : 300' (in a text box) and a 'Take Money from savings : ' button. The text 'monthly input to our savings :' is positioned above the '300' input field.

Field	Value
Username	pawel
Expenses	399
Balance	900677
Monthly Salary	200
Recipient's username	ahmad
Amount	20000
monthly input to our savings	300
SAVING BALANCE	9962.01
FINAL USABLE MONEY	29155.7

And for the investment account we have the following :

Investment Account Window

BUY AND SELL STOCKS :

Tab 1 Tab 2

CHOOSE AND BUY

Select Stock :
APPL

Select the number of share :

BUY

Owned Stocks :

AAPL	0
MSFT	20
TSLA	20
AMZN	20
GOOGL	100

Username : yy

Balance : 705.246

Recipient's username :

Transfer

Amount :

Log out

MSFT current share value : 100.231

You can buy stocks, which their share values are being randomized according to the time, and sell them afterwards.

Internal specification :

(function headers and what do the functions do, and how they should be used, list of parameters and their roles, results)

In the Bank class we have two main functionalities :

1) update stock function :

```
void updateStockValue(Stock stock) {
    // todo fix this
    float shareFraction = stock.share / 5;

    while (1) { // CAUTION: This is an infinite loop. Must break from the loop.
        clock_t this_time = clock();
        clock_t last_time = this_time;

        if (this_time - last_time > 10) {
            stock.share += (
#ifdef __APPLE__ // Preprocessor to check which OS we are using and provide code based on that
                arc4random()
            else
                random() // We simply can use random() function here but I hav
            ) * shareFraction;
            // a.value = a.volume * a.share;
        }
    };
}
```

2) saving all the accounts from a file :

```
void saveAccounts() {
    saveAccounts(bankFile);
}

void saveAccounts(const string &filename) {          // save all the contents of a map in a file

    QFile bankFile(QString::fromStdString(filename)); // Get the std::string to QString then open a QFile

    if (bankFile.open(QFile::ReadWrite)) { // Open the file to ReadWrite into it and clear the old content of the file
        QTextStream out(&bankFile); //<--QT text for file handling-- Create a Text stream to write into the file

        std::stringstream ss; // Used to get the std::ostream output of our NormalAccount

        for (auto it = accounts.begin(); it != accounts.end(); it++) {
            ss << *(it->second);
        }

        out << QString::fromStdString(ss.str()); // Carver the std::stringstream to QString and write into the file

        bankFile.close(); // Close the file
    } else {
        std::cout << "Can not open file " << filename << " to save account." << endl;
    }
}
```

3) in the mainWindow

```
QMainWindow *w = nullptr;

if (username.size() && password.size() != 0) { // If username & password are not empty not empty

    NormalAccount *account = bank->getAccount(username); // Get the account class from the bank account map

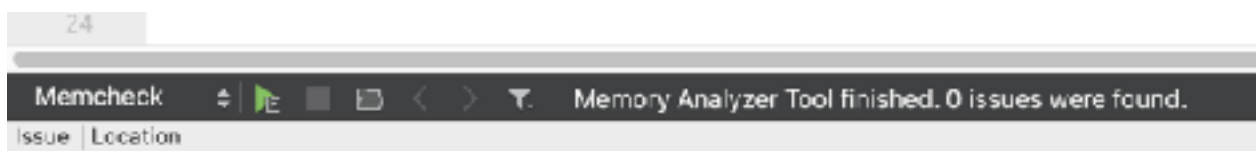
    // Check the account type and open appropriate window

    // UPDATED
    // Giving the parent reference so that whenever MainWindow is deleted it's child will also be deleted. (A Qt parent child fun
    if (account->getAccountType() == Normal) {
        w = new NormalWindow(account, bank, this);
    } else if (account->getAccountType() == Investment) {
        w = new InvestmentWindow(dynamic_cast<InvestmentAccount*>(account), bank, this);
    } else if (account->getAccountType() == Savings) {
        w = new SavingWindow(dynamic_cast<SavingAccount*>(account), bank, this);
    }
}

if (w) {
    w->show(); // Show the window if it is not nullptr
}
}
```


Testing :

Concerning the memory leaks, since we are using the IDE from the QT library where it is easier to manage both your designs and code, we have the ability to test our memory leaks without the need of external libraries.



After getting the coding done, we tested all the possibilities from our program :

Topics :

The topics from the laboratories which are used in this program are :

- 1) streams : (to read and write the accounts which are present in the bank, or the ones which are being created)

```
QFile bankFile(QString::fromStdString(filename)); // Get the std::string to QString then open a QFile

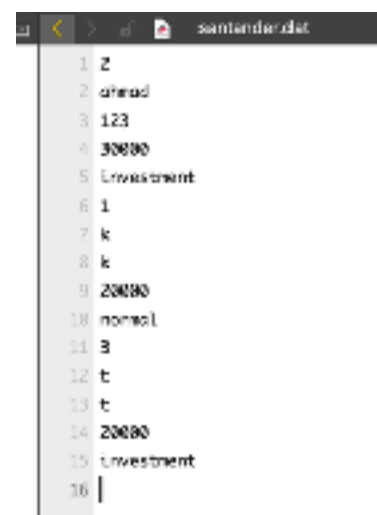
if (bankFile.open(QFile::ReadWrite)) { // Open the file to ReadWrite into it and clear the old content of the file
    QTextStream out(&bankFile); //<--QT text for file handling-- Create a Text stream to write into the file

    std::stringstream ss; // Used to get the std::ostream output of our NormalAccount

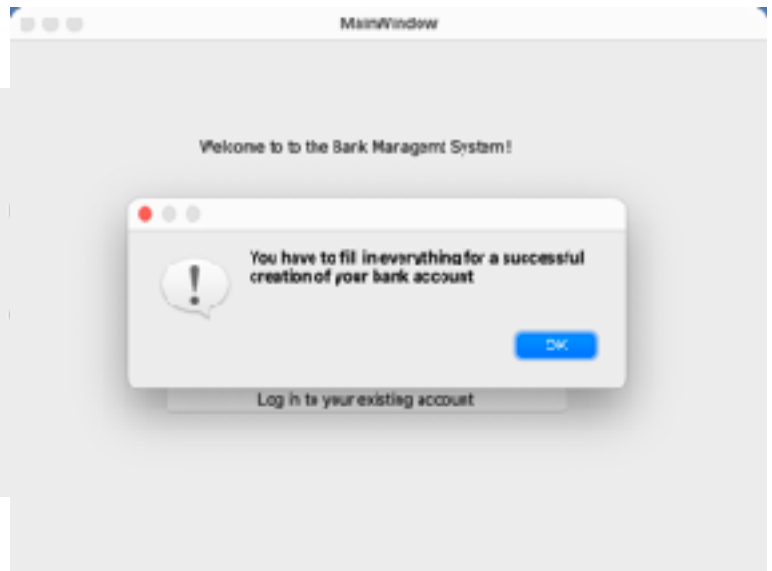
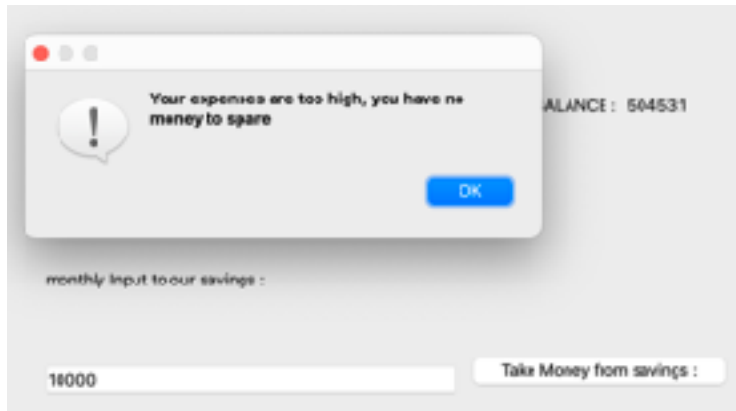
    for (auto it = accounts.begin(); it != accounts.end(); it++) {
        ss << *(it->second);
    }

    out << QString::fromStdString(ss.str()); // Convert the std::stringstream to QString and write into the file

    bankFile.close(); // Close the file
} else {
    std::cout << "Can not open file " << filename << " to save account." << endl;
}
```



- 2) Exceptions : handling the cases where : -the username wants to create an account with an already existing username, - the user has not filled all the data for the successful creation of the account, or for exemple when a user wants to buy a number of shares of a stock, but doesn't has sufficient balance to do so, and so on.



- 3) Inheritance : Both the investment account and the saving account are inherited from the class NormalAccount, we used this concept since all of the account have similar core structure and functionalities.
- 4) Polymorphism : in order to show the correct window after log in, polymorphism was used in order to return the correct data type of the account from one single virtual function "getAccount"

```

QMainWindow *w = nullptr;

if (username.size() && password.size() != 0) { // If username & password are not empty

    NormalAccount *account = bank->getAccount(username); // Get the account class from the bank account map

    // Check the account type and open appropriate window
    // Giving the parent reference so that whenever QMainWindow is deleted it's child will also be deleted. (A Qt parent child functionality)
    if (account->getAccountType() == Normal) {
        w = new NormalWindow(account, bank, this);
    } else if (account->getAccountType() == Investment) {
        w = new InvestmentWindow(dynamic_cast<InvestmentAccount*>(account), bank, this);
    } else if (account->getAccountType() == Savings) {
        w = new SavingWindow(dynamic_cast<SavingsAccount*>(account), bank, this);
    }
}

if (w) {
    w->show(); // Show the window if it is not nullptr
}
}

```

As you can see after checking the results of each account type, a new QMainWindow object (w) will be created, pointing at the correct account window, and it is getting executed after we get the correct account type from the user.

Conclusions :

Learning to code such program, will give any programmer a good understanding of the Qt library and how it can be used for different graphical user interfaces development while enforcing their knowledge in c++.

Qt is one of the libraries which are the most used for graphical user interface programming, since it is well documented and easy to install and run on every platform, which was not the case with my recent trial with wxWidgets.

This program can be used to simplify the way a modern bank may handle their data to run their institution, and give an understanding on how buying and selling stock markets work.

References used in order to achieve this project :

Lectures slides from our course.

cplusplus.com

geekforgeek.com

doc.qt.io

stackoverflow.com

Qt YouTube channel for design tips and code integration.