**Ministry of education and science of the Kyrgyz Republic**

**Kyrgyz State Technical University named after I.Razzakov**

**Institute of Information Technologies**

**Department of Software of Computer Systems**

**Major: 710400 «Software Engineering»**

Report

Discipline: «**Object-Oriented Design lab work 12**»

Software requirements

Done by**:** student of the SE(eng)- 1- 21

Shumkarbekov Sultan

Checked by: Musabaev E.B.

Bishkek 2024

Table of contents

[Introduction 3](#_Toc165016603)

[Functional requirements: 3](#_Toc165016604)

[Non-functional requirements: 3](#_Toc165016605)

[Code 4](#_Toc165016606)

# Introduction

The Commercial Banks Directory application is designed to manage information about commercial banks, including their names, addresses, ownership forms, and terms for storing funds in personal accounts (annual interest rates on various types of deposits). The application allows users to add new banks, save and load the directory from a file, view the list of banks with their details, and select the bank with the highest interest rate for a specified type of deposit. The application provides a user-friendly interface for efficient management of bank information, ensuring reliability, security, and scalability.

# Functional requirements:

Adding a bank: The user can add a new bank to the directory by specifying its name, address, form of ownership and information about deposits (deposit name and annual interest rate).

Saving the directory: The user can save the directory of banks to a file for later use.

Downloading the directory: The user can download an existing directory of banks from a file.

Viewing the list of banks: The user can view a list of all banks in the directory, including information about their name, address, form of ownership and deposits.

Selecting the bank with the highest interest rate: The user can select the bank with the highest interest rate for a given type of deposit.

# Non-functional requirements:

Performance: When adding, saving and loading banks, the directory must work quickly and efficiently, even with a large number of records.

User-friendliness: The application interface should be intuitive and easy to use, even for users without special skills.

Reliability: The application must be stable and reliable, without the possibility of data loss in unforeseen situations.

Security: Access to bank and deposit data must be secured to prevent unauthorized access or modification of data.

Scalability: The application should be easily scalable to add new features and support more data in the future.

# Code

#include <sstream>

#include <vector>

#include <string>

#include <algorithm>

class Bank {

public:

std::string name;

std::string address;

std::string ownership;

std::vector<std::pair<std::string, double>> deposits; // pair<deposit\_name, annual\_interest\_rate>

Bank(const std::string& n, const std::string& addr, const std::string& own)

: name(n), address(addr), ownership(own) {}

void addDeposit(const std::string& depositName, double annualInterestRate) {

deposits.push\_back({ depositName, annualInterestRate });

}

std::string toString() const {

std::stringstream ss;

ss << "Bank Name: " << name << "\n";

ss << "Address: " << address << "\n";

ss << "Ownership: " << ownership << "\n";

ss << "Deposits:\n";

for (const auto& deposit : deposits) {

ss << " - " << deposit.first << " (Annual Interest Rate: " << deposit.second << "%)\n";

}

return ss.str();

}

};

class BankDirectory {

private:

std::vector<Bank> banks;

public:

void addBank(const Bank& bank) {

banks.push\_back(bank);

}

void saveToFile(const std::string& filename) const {

std::ofstream outFile(filename);

if (!outFile.is\_open()) {

return;

}

for (const auto& bank : banks) {

outFile << bank.name << "," << bank.address << "," << bank.ownership;

for (const auto& deposit : bank.deposits) {

outFile << "," << deposit.first << ":" << deposit.second;

}

outFile << std::endl;

}

outFile.close();

}

void loadFromFile(const std::string& filename) {

std::ifstream inFile(filename);

if (!inFile.is\_open()) {

return;

}

banks.clear(); // Clear existing data before loading new data

std::string line;

while (std::getline(inFile, line)) {

std::istringstream iss(line);

std::string name, address, ownership, depositInfo;

getline(iss, name, ',');

getline(iss, address, ',');

getline(iss, ownership, ',');

Bank bank(name, address, ownership);

while (getline(iss, depositInfo, ',')) {

size\_t pos = depositInfo.find(':');

if (pos != std::string::npos) {

std::string depositName = depositInfo.substr(0, pos);

double annualInterestRate = std::stod(depositInfo.substr(pos + 1));

bank.addDeposit(depositName, annualInterestRate); } }

addBank(bank); }

inFile.close(); }

std::vector<Bank> getBanks() const {

return banks; }};

class BankApp : public QWidget {

Q\_OBJECT

public:

BankApp(QWidget\* parent = nullptr)

: QWidget(parent) {

QVBoxLayout\* layout = new QVBoxLayout(this);

listWidget = new QListWidget(this);

layout->addWidget(listWidget);

addButton = new QPushButton("Add Bank", this);

connect(addButton, &QPushButton::clicked, this, &BankApp::addBankClicked);

layout->addWidget(addButton);

saveButton = new QPushButton("Save Banks", this);

connect(saveButton, &QPushButton::clicked, this, &BankApp::saveBanksClicked);

layout->addWidget(saveButton);

loadButton = new QPushButton("Load Banks", this);

connect(loadButton, &QPushButton::clicked, this, &BankApp::loadBanksClicked);

layout->addWidget(loadButton);

setLayout(layout); }

private slots:

void addBankClicked() {

// Implement adding a bank }

void saveBanksClicked() {

// Implement saving banks to a file }

void loadBanksClicked() {

// Implement loading banks from a file }

private:

QListWidget\* listWidget;

QPushButton\* addButton;

QPushButton\* saveButton;

QPushButton\* loadButton;

BankDirectory directory;

};

int main(int argc, char\* argv[]) {

QApplication app(argc, argv);

BankApp bankApp;

bankApp.show();

return app.exec();

}

#include "main.moc"