**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**

**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Проектний Практикум**

**Практична робота №1**

**на тему: «Проектування і реалізація програми з ієрархієб класів (одиночне і множинне успадкування»**

Виконав: студент 3 курсу, групи ІПЗ-3.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Перевірив\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Багачук Д.Г.

**Одеса  2023**

**МЕТА РОБОТИ**

Засвоєння поняття успадкування та його принципів; набуття практичних навичок з оголошення та використання ієрархій класів.

**ЗАВДАННЯ**

**Основні завдання роботи:**

Розробити та реалізувати програмно ієрархі класів для роботи з рахунком вкладника банку для цього слід виконати такі завдання

1. Розробити та реалізувати програмно базовий клас
2. Розробити та реалізувати програмно базовий клас
3. Розробити та реалізувати програмно похідний клас головній частині програми продемонструвати роботу створено ієрархі класів

Порядок виконання роботи

1. Створити у програмі клас Rahunok, який зберігатиме такі дані: унікальний номер рахунка суму на рахунку початкове значення задати при створенні рахунка нарахування відсотків
2. Додати до класу функції: конструктора з параметрами, відображення рахунка, встановлення додаткових відсотків, перерахування грошей з урахуванням відсотків
3. Створити у програмі клас Vkladnyk, який зберігатиме такі дані: серію та номер паспорту, прізвище, ім я та по батькові.
4. Додати до класу конструктора з параметрами функцію відображення даних про вкладника.
5. Створити у програмі похідний клас RahunokVkladnyka від попередніх двох класів. Цей клас має містити дані про максимальну суму яку можна зняти за добу, мінімальний залишок на рахунку після зняття грошей з рахунка, пароль, а також функції: конструктор з параметрами, функцію виведення інформації про рахунок вкладника, функцію зміни пароля, функцію вкладення грошей, функцію зняття грошей. Врахуйте необхідну політику конфіденційності та безпеки даних банку.
6. У головній частині програми продемонструйте роботу створеної ієрархії класів на прикладі такого об єкта

RahunokVkladnyka Geits(123456789, “Bill Geits”, “EC123456”, 50000, 500, 500);

**Код програми:**

**main.cpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #6; \**

*\* Student Bukhta Mykyta; \**

*\* Grade: 3; \**

*\* Group Software Engineering 3.04; \* \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#include "BankAccount.hpp"

#include "Owner.hpp"

#include "BankAccountOwner.hpp"

#include <iostream>

#include <string>

using namespace lab\_6;

std::string get\_withdraw\_output\_information(BankAccountOwner &owner, uint64\_t sum) {

std::string ret;

if (!owner.withdraw(sum)) {

auto max\_withdraw\_sum = owner.max\_witdraw\_sum();

auto min\_balance\_sum = owner.min\_balance();

auto balance = owner.balance();

ret = std::string{"Warning! You want to withdraw more than you can do by your account limit!\nWithdraw sum: "}

+ std::to\_string(sum) + "\nCurrent balance: " + std::to\_string(balance)

+ "\nMax withdraw daily sum: " + std::to\_string(max\_withdraw\_sum)

+ "\nMin sum should be left on the account: " + std::to\_string(min\_balance\_sum);

} else {

ret = "Operation successfull!";

}

return ret;

}

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

Owner::Passport init\_passport{Owner::FullName{"Bill", "Gates"}, "1234", "123456789"};

Owner owner{init\_passport};

BankAccount bank\_account(30000, 300);

BankAccountOwner balance\_account\_owner(owner, bank\_account, 50000, 500, "EC123456");

std::cout << bank\_account.convert\_to\_floating\_string(10000) << "\n\n";

std::cout << owner.to\_string() << "\n\n";

std::cout << balance\_account\_owner.to\_string() << "\n\n";

std::cout << get\_withdraw\_output\_information(balance\_account\_owner, 50100) << "\n\n";

std::cout << get\_withdraw\_output\_information(balance\_account\_owner, 40000) << "\n\n";

std::cout << get\_withdraw\_output\_information(balance\_account\_owner, 30000) << "\n\nB";

std::cout << get\_withdraw\_output\_information(balance\_account\_owner, 25000) << std::endl;

return 0;

}

**BankAccount.hpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #6; \**

*\* Student Bukhta Mykyta; \**

*\* Grade: 3; \**

*\* Group Software Engineering 3.04; \* \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNT\_HPP

#define BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNT\_HPP

#include <inttypes.h>

#include <string>

namespace lab\_6 {

class BankAccount {

public:

BankAccount(void);

BankAccount(uint64\_t balance, uint16\_t interest\_rate);

virtual ~BankAccount(void) = default;

*/\**

*\* Getters / setters starts;*

*\*/*

uint64\_t balance(void) const noexcept;

uint16\_t interest\_rate(void) const noexcept;

void set\_interest\_rate(uint16\_t val) noexcept;

*/\**

*\* Getters / setters ends;*

*\*/*

uint64\_t calculate\_balance\_with\_interest(void) const noexcept;

*/\**

*\* Static methods starts;*

*\*/*

*/\**

*\* Class interface starts;*

*\*/*

std::string to\_string(void) const;

*/\**

*\* Class interface ends;*

*\*/*

*/\* Description:*

*\* Convert int number to float as std::string;*

*\*/*

static std::string convert\_to\_floating\_string(uint64\_t val);

*/\**

*\* Static methods ends;*

*\*/*

protected:

uint64\_t m\_balance;

private:

uint32\_t m\_id;

static uint32\_t m\_id\_counter;

uint16\_t m\_interest\_rate;

};

} *// !lab\_6;*

#endif *// !BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNT\_HPP;*

**BankAccount.cpp**

#include "BankAccount.hpp"

#include "Config.hpp"

namespace lab\_6 {

uint32\_t BankAccount::m\_id\_counter = 0;

BankAccount::BankAccount(void) : BankAccount(0, 0) {

}

BankAccount::BankAccount(uint64\_t balance, uint16\_t interest\_rate)

: m\_balance{balance}, m\_interest\_rate{interest\_rate}

{

m\_id = m\_id\_counter++;

}

*/\**

*\* Getters / setters starts;*

*\*/*

uint64\_t BankAccount::balance(void) const noexcept {

return m\_balance;

}

uint16\_t BankAccount::interest\_rate(void) const noexcept {

return m\_interest\_rate;

}

void BankAccount::set\_interest\_rate(uint16\_t val) noexcept {

m\_interest\_rate = val;

}

*/\**

*\* Getters / setters ends;*

*\*/*

uint64\_t BankAccount::calculate\_balance\_with\_interest(void) const noexcept {

return m\_balance / config::FULL\_PERCENT \* m\_interest\_rate / config::NUMBER\_CONVERTOR\_DEVIDER + m\_balance;

}

std::string BankAccount::to\_string(void) const {

std::string ret{std::move(convert\_to\_floating\_string(m\_balance))};

return std::move(ret);

}

*/\**

*\* Static methods starts;*

*\*/*

std::string BankAccount::convert\_to\_floating\_string(uint64\_t val) {

std::string ret{std::move(std::to\_string(val))};

ret.insert(ret.end() - config::FLOATING\_NUMBERS\_COUNT, config::FLOATING\_DELIMITER);

return std::move(ret);

}

*/\**

*\* Static methods ends;*

*\*/*

} *// !lab\_6;*

**Owner.hpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #6; \**

*\* Student Bukhta Mykyta; \**

*\* Grade: 3; \**

*\* Group Software Engineering 3.04; \* \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_6\_OWNER\_HPP

#define BUKHTAMYKYTA\_LAB\_6\_OWNER\_HPP

#include <string>

namespace lab\_6 {

class Owner {

public:

struct FullName {

std::string first\_name;

std::string second\_name;

std::string middle\_name;

FullName(void) = default;

FullName(std::string init\_first\_name, std::string init\_second\_name, std::string init\_middle\_name = "");

bool operator== (const FullName &other) const;

bool operator!= (const FullName &other) const;

std::string to\_string(void) const;

};

struct Passport {

std::string series;

std::string number;

FullName full\_name;

Passport(void) = default;

Passport(FullName init\_full\_name, std::string init\_series, std::string init\_number);

bool operator== (const Passport &other) const;

bool operator!= (const Passport &other) const;

std::string to\_string(void) const;

};

public:

Owner(void) = default;

Owner(const Passport &passport);

virtual ~Owner() = default;

*/\**

*\* Getters / setters starts;*

*\*/*

Passport passport(void) const noexcept;

*/\**

*\* Getters / setters ends;*

*\*/*

*/\**

*\* Class interface starts;*

*\*/*

std::string to\_string(void) const;

*/\**

*\* Class interface ends;*

*\*/*

private:

Passport m\_passport;

};

} *// !lab\_6;*

#endif *// !BUKHTAMYKYTA\_LAB\_6\_OWNER\_HPP;*

**Owner.cpp**

#include "Owner.hpp"

#include "Config.hpp"

#include <array>

#include <algorithm>

#include <numeric>

namespace lab\_6 {

*/\**

*\* Struct FullNmae starts;*

*\*/*

Owner::FullName::FullName(std::string init\_first\_name, std::string init\_second\_name, std::string init\_middle\_name)

: first\_name{init\_first\_name}, second\_name{init\_second\_name}, middle\_name{init\_middle\_name}

{

}

bool Owner::FullName::operator== (const FullName &other) const {

return this->middle\_name == other.middle\_name && this->second\_name == other.second\_name && this->first\_name == other.first\_name;

}

bool Owner::FullName::operator!= (const FullName &other) const {

return !this->operator==(other);

}

std::string Owner::FullName::to\_string(void) const {

*// 3 - is a count of the string vars. We need it to add the ' ' to the string;*

std::array<const std::string\*, 3> init\_list = {&second\_name, &first\_name, &middle\_name};

std::string ret = std::move(std::accumulate(init\_list.begin(), init\_list.end(), std::string{}, [](std::string &sum, const std::string \*elem) {

if (!sum.empty()) {

sum.push\_back(' ');

}

return std::move(sum + \*elem);

}));

return std::move(ret);

}

*/\**

*\* Struct FullNmae ends;*

*\*/*

Owner::Passport::Passport(FullName init\_full\_name, std::string init\_series, std::string init\_number)

: full\_name{init\_full\_name}, series{init\_series}, number{init\_number}

{

}

bool Owner::Passport::operator== (const Passport &other) const

{

return this->series == other.series && this->number == other.number;

}

std::string Owner::Passport::to\_string(void) const {

std::string string\_full\_name{std::move(full\_name.to\_string())};

std::array<const std::string\*, 3> init\_list = {&string\_full\_name, &series, &number};

std::string ret = std::move(std::accumulate(init\_list.begin(), init\_list.end(), std::string{}, [](std::string &sum, const std::string \*elem) {

sum.push\_back(' ');

return std::move(sum + \*elem);

}));

return std::move(ret);

}

Owner::Owner(const Passport &passport)

: m\_passport(passport)

{

}

*/\**

*\* Getters / setters starts;*

*\*/*

Owner::Passport Owner::passport(void) const noexcept

{

return m\_passport;

}

*/\**

*\* Getters / setters ends;*

*\*/*

std::string Owner::to\_string(void) const

{

return std::move(m\_passport.to\_string());

}

} *// !lab\_6;*

**BankAccountOwner.hpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #6; \**

*\* Student Bukhta Mykyta; \**

*\* Grade: 3; \**

*\* Group Software Engineering 3.04; \* \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNTOWNER\_HPP

#define BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNTOWNER\_HPP

#include "Owner.hpp"

#include "BankAccount.hpp"

#include <inttypes.h>

#include <string>

namespace lab\_6 {

class BankAccountOwner : public Owner, public BankAccount {

public:

*// BankAccountOwner(void) = default;*

BankAccountOwner(uint64\_t max\_withdraw\_sum, uint64\_t min\_balance, const std::string &password);

BankAccountOwner(const Owner::Passport &m\_passport, uint64\_t balance, uint16\_t interest\_rate, uint64\_t max\_withdraw\_sum,

uint64\_t min\_balance, const std::string &password);

BankAccountOwner(const Owner &owner, const BankAccount &account, uint64\_t max\_withdraw\_sum, uint64\_t min\_balance, const std::string &password);

virtual ~BankAccountOwner(void) = default;

*/\**

*\* Getters/Setters starts;*

*\*/*

uint64\_t max\_witdraw\_sum(void) const noexcept;

uint64\_t min\_balance(void) const noexcept;

void set\_password(const std::string &new\_password);

*/\**

*\* Getters/Setters ends;*

*\*/*

*/\**

*\* Class interface starts;*

*\*/*

std::string to\_string(void) const;

void cash\_in(uint64\_t sum) noexcept;

*/\* Description:*

*\* Method change the balance of the owner account. We cannot to withdraw more money,*

*\* then the balance could have after operation. The balance should have more than m\_MIN\_BALANCE*

*\* money for this operation or not more than daily limit (m\_MAX\_WITHDRAW\_SUM);*

*\**

*\* Args:*

*\* sum - count of money user want to withdraw. The last 2 numbers are kopecs value.*

*\**

*\* Return value.*

*\* true - operation was successfull.*

*\* false - sum is not valid;*

*\*/*

bool withdraw(uint64\_t sum) noexcept;

*/\**

*\* Class interface ends;*

*\*/*

private:

const uint64\_t m\_MAX\_WITHDRAW\_SUM;

const uint64\_t m\_MIN\_BALANCE;

std::string m\_password;

};

} *// !lab\_6;*

#endif *// !BUKHTAMYKYTA\_LAB\_6\_BANKACCOUNTOWNER\_HPP;*

**BankAccountOwner.cpp**

#include "BankAccountOwner.hpp"

#include "Config.hpp"

namespace lab\_6 {

BankAccountOwner::BankAccountOwner(uint64\_t max\_withdraw\_sum, uint64\_t min\_balance, const std::string &password)

: m\_MAX\_WITHDRAW\_SUM{max\_withdraw\_sum}, m\_MIN\_BALANCE{min\_balance}

{

}

BankAccountOwner::BankAccountOwner(const Owner::Passport &passport, uint64\_t balance, uint16\_t interest\_rate, uint64\_t max\_withdraw\_sum,

uint64\_t min\_balance, const std::string &password)

: BankAccountOwner(Owner{passport}, BankAccount{balance, interest\_rate}, max\_withdraw\_sum, min\_balance, password)

{

}

BankAccountOwner::BankAccountOwner(const Owner &owner, const BankAccount &account, uint64\_t max\_withdraw\_sum, uint64\_t min\_balance, const std::string &password)

: Owner{owner}, BankAccount(account), m\_MAX\_WITHDRAW\_SUM{max\_withdraw\_sum}, m\_MIN\_BALANCE{min\_balance}, m\_password{password}

{

}

*/\**

*\* Getters/Setters starts;*

*\*/*

uint64\_t BankAccountOwner::max\_witdraw\_sum(void) const noexcept {

return m\_MAX\_WITHDRAW\_SUM;

}

uint64\_t BankAccountOwner::min\_balance(void) const noexcept {

return m\_MIN\_BALANCE;

}

void BankAccountOwner::set\_password(const std::string &val) {

m\_password = val;

}

*/\**

*\* Getters/Setters starts;*

*\*/*

*/\**

*\* Class interface starts;*

*\*/*

std::string BankAccountOwner::to\_string(void) const {

return std::move(Owner::to\_string() + config::DATA\_DELIMITER\_STRING + BankAccount::to\_string());

}

void BankAccountOwner::cash\_in(uint64\_t sum) noexcept {

m\_balance += sum;

}

bool BankAccountOwner::withdraw(uint64\_t sum) noexcept {

if (sum > m\_MAX\_WITHDRAW\_SUM || sum > m\_balance || (m\_balance - sum) < m\_MIN\_BALANCE) {

return false;

}

m\_balance -= sum;

return true;

}

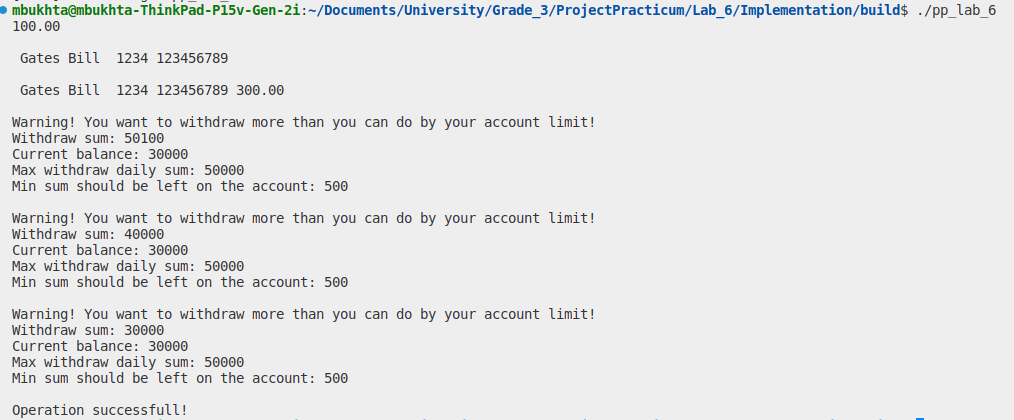
*/\**

*\* Class interface ends;*

*\*/*

} *// !lab\_6;*

**Результат виконання програми:**



**ВИСНОВОК:**

Під час виконання цієї роботи ми успішно оволоділи поняттям успадкування та його ключовими принципами в об'єктно-орієнтованому програмуванні. Ми також набули важливих практичних навичок в області оголошення та використання ієрархій класів.

Розуміння успадкування є фундаментальним для розроблення програм із багаторазовим використанням коду та створення більш зручних і ефективних структур. Ми засвоїли, як створювати похідні класи, що успадковують властивості та методи від базових класів, і застосовувати успадкування для вирішення практичних завдань.

Ці нові знання та навички будуть корисними при розробці програмних проєктів, де потрібне створення ієрархій класів, а також при оптимізації та підтримці наявного коду.