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;
```

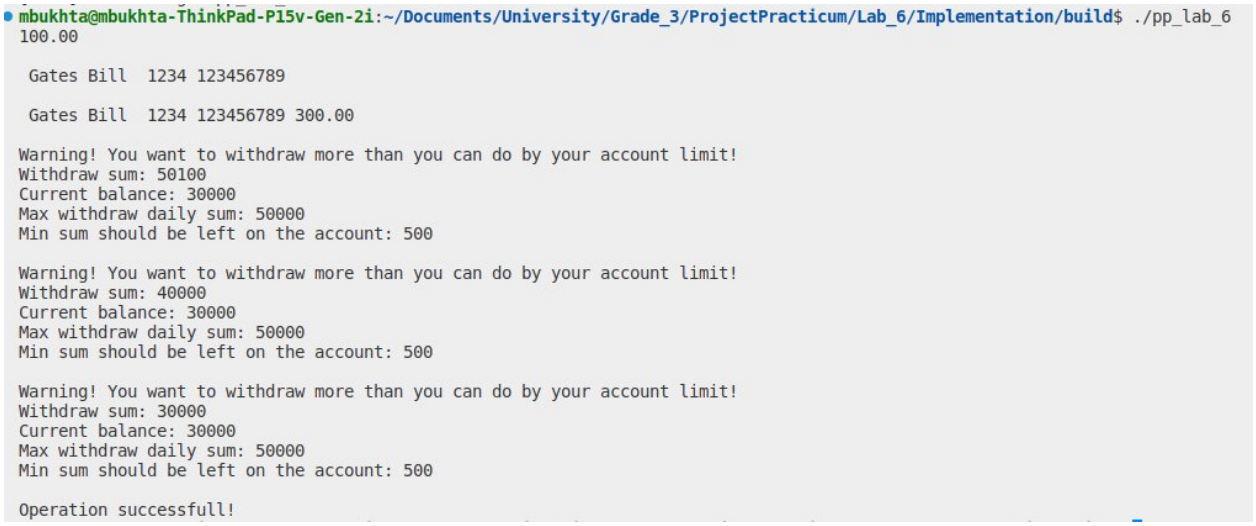
```
}

/*
 * 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.