

**МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)  
Кафедра САПР**

**ОТЧЕТ  
по лабораторной работе №6  
по дисциплине «Объектно-ориентированное  
программирование»  
Тема: «Создание и использование классов»**

Студенты гр. 1335

\_\_\_\_\_ Максимов Ю. Е.

Преподаватель:

\_\_\_\_\_ Новакова Н.Е.

Санкт-Петербург

2024

## **1. Цель работы**

Изучение классов и инкапсуляции в языке C++ с помощью программного продукта компании CLion.

## **2. Анализ задачи**

Необходимо:

- 1) Написать программу, которая преобразовывает структуру, описывающую банковский счет в класс;
- 2) Написать программу, которая вносит изменения в прошлую программу, добавляет автоматическое назначение номера аккаунта;
- 3) Написать программу, которая вносит изменения в прошлую программу, добавляет два метода, которые добавляет и снимает деньги.

## **3. Ход выполнения работы**

### **3.1 Упражнение 1**

В ходе выполнения данного упражнения написана программа, которая преобразует структуру в класс, описывающий банковский счет с параметрами: номер, счет и тип аккаунта.

#### **3.1.1 Пошаговое описание алгоритма**

- 1) Изучить данные предоставленных файлов.
- 2) Изменить в BankAccount.cs структуру на класс.
- 3) Добавить метод Populate для назначения.
- 4) Добавить три метода, которые возвращают значения трех полей.
- 5) Изменить метод Write.
- 6) Ввод номера, счета.
- 7) Вывести на экран номер, счет и тип аккаунтов.

### 3.1.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- `std::cout` – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;
- `std::cin` – ожидает следующего нажатия клавиши пользователем;
- `main()` – служит для запуска программы.
- `BankAccount` – класс, который описывает банковский аккаунт с помощью методов: `Populate` (создание), `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта).
- `CreateAccount` – класс, который содержит методы: `NewBankAccount` (создание аккаунта), `Write` (вывод информации об аккаунте).

### 3.1.3 Контрольный пример

На рис.3.1 представлены результаты выполнения программы.

```
"C:\Users\dokto\OneDrive\Рабочий стол\learn\oop\src\lab6\task10\cmake-build-debug\task10.exe"
Enter account number:1
Enter the account balance! :100
Account number is 1
Account balance is 100
Account type is Checking

Process finished with exit code 0
|
```

Рис.3.1 Контрольный пример для программы

Как видно из рисунка, пользователем вводятся номера и баланс аккаунтов и на экран выводятся результаты.

## **3.2 Упражнение 2**

В ходе выполнения данного упражнения, написана программа, которая вносит изменения в прошлую программу, добавляет автоматическое назначение номера аккаунта.

### **3.2.1 Пошаговое описание алгоритма**

- 1) Ввод счета.
- 2) Использование метода `Populate`.
- 3) Вывод на экран номера, счета и типов аккаунтов.

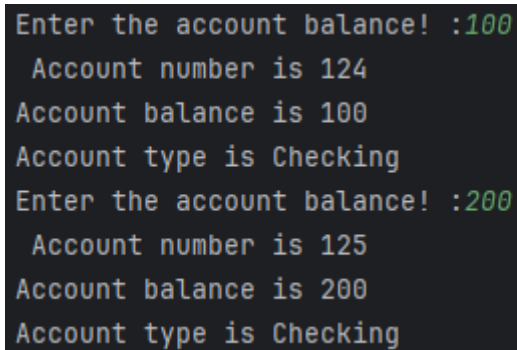
### **3.2.2 Используемые классы и методы**

В программе, написанной в данном упражнении, используются следующие методы:

- `System.Console.WriteLine()` – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;
- `System.Console.ReadKey()` – ожидает следующего нажатия клавиши пользователем;
- `Main()` – служит для запуска программы.
- `BankAccount` – класс, который описывает банковский аккаунт с помощью методов: `Populate` (создание), `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта), `NextNumber` (считает номер аккаунта при создании).
- `CreateAccount` – класс, который содержит методы: `NewBankAccount` (создание аккаунта), `Write` (вывод информации об аккаунте).

### **3.2.3 Контрольный пример**

На рис.3.2 представлены результаты выполнения программы.



```
Enter the account balance! :100
Account number is 124
Account balance is 100
Account type is Checking
Enter the account balance! :200
Account number is 125
Account balance is 200
Account type is Checking
```

Рис.3.2 Контрольный пример для программы

Как видно из рисунка, пользователем вводятся балансы аккаунтов и на экран выводятся результаты.

### 3.3 Упражнение 3

В ходе выполнения данного упражнения, написана программа, которая вносит изменения в прошлую программу, добавляет два метода, один вносит деньги на счет, другой снимает деньги со счета.

#### 3.3.1 Пошаговое описание алгоритма

- 1) Ввод счета.
- 2) Использование метода Populate.
- 3) Использование методов TestDeposit и TestWithdraw, которые вносят деньги на счет и снимают их.
- 4) Вывод на экран номера, счета и типов аккаунтов.

#### 3.3.2 Используемые классы и методы

В программе, написанной в данном упражнении, используются следующие методы:

- `System.Console.WriteLine()` – служит для отображения на экране строк и значений переменных, переданных в метод в качестве параметров, с переходом на новую строку;

- `System.Console.ReadKey()` – ожидает следующего нажатия клавиши пользователем;

- `Main()` – служит для запуска программы.

- `BankAccount` – класс, который описывает банковский аккаунт с помощью методов: `Populate` (создание), `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта), `NextNumber` (считает номер аккаунта при создании), `Withdraw` (позволяет произвести снятие денег, только в 3 пункте) и `Deposit` (позволяет пополнить счет (только в 3 пункте)).

- `CreateAccount` – класс, который содержит методы: `NewBankAccount` (создание аккаунта), `Write` (вывод информации об аккаунте), `TestWithdraw` (попытка снятия денег), `TestDeposit` (попытка внесения денег).

### 3.3.3 Контрольный пример

На рис.3.3 представлены результаты выполнения программы.

```
"C:\Users\dokto\OneDrive\Рабочий стол\learn\oop\src\lab6\task12\cmake-build-debug\task12.exe"
Enter the account balance! :100
Account number is 124
Account balance is 100
Account type is Checking
Account number is 124
Account balance is 110
Account type is Checking
Account number is 124
Account balance is 60
Account type is Checking

Enter the account balance! :200\
Account number is 125
Account balance is 200
Account type is Checking
Account number is 125
Account balance is 210
Account type is Checking
Account number is 125
Account balance is 160
Account type is Checking

Process finished with exit code 0
```

Рис.3.3 Контрольный пример для программы

Как видно из рисунка, пользователем вводятся данные для работы программы и на экран выведены результаты.

#### 4. Листинг программы

##### **AccountType.h**

```
//  
  
// Created by dokto on 11.06.2024.  
  
//  
  
#pragma once  
  
namespace bank {  
  
    enum class AccountType {  
  
        Checking,  
  
        Deposit  
  
    };  
  
}
```

##### **bankAccount.cpp**

```
//  
  
// Created by dokto on 11.06.2024.  
  
//  
  
#include "bankAccount.h"  
  
/**  
  
 * Populates the bank account with the given account number and balance.  
  
 *  
  
 * @param number The account number.  
  
 * @param balance The account balance.
```

\*

\* @throws None.

\*/

```
void bank::BankAccount::populate(const int number, const double balance) {
```

```
    accNo = number;
```

```
    accBal = balance;
```

```
    accType = AccountType::Checking;
```

```
}
```

/\*\*

\* Returns the account number.

\*

\* @return The account number.

\*

\* @throws None.

\*/

```
auto bank::BankAccount::getNumber() const -> long {
```

```
    return accNo;
```

```
}
```

/\*\*

\* Returns the account balance.

\*

\* @return The account balance.

\*

\* @throws None.

\*/



```
auto bank::BankAccount::getBalance() const -> double {  
  
    return accBal;  
  
}
```

```
/**
```

```
 * Returns the account type.
```

```
 *
```

```
 * @return The account type.
```

```
 *
```

```
 * @throws None.
```

```
 */
```

```
auto bank::BankAccount::getType() const -> AccountType {  
  
    return accType;  
  
}
```

```
/**
```

```
 * Sets the account number.
```

```
 *
```

```
 * @param number The account number.
```

```
 *
```

```
 * @throws None.
```

```
 */
```

```
auto bank::BankAccount::setNumber(const long number) -> void {  
  
    accNo = number;  
  
}
```

```
/**
```

\* Sets the balance of the bank account.

\*

\* @param balance The new balance to be set.

\*

\* @return void

\*

\* @throws None

\*/

```
auto bank::BankAccount::setBalance(const double balance) -> void {
```

```
    accBal = balance;
```

```
}
```

/\*\*

\* Sets the account type.

\*

\* @param type The new account type.

\*

\* @return void

\*

\* @throws None

\*/

```
auto bank::BankAccount::setType(const AccountType type) -> void {
```

```
    accType = type;
```

```
}
```

## **bankAccount.h**

//

// Created by dokto on 11.06.2024.

//

```

#pragma once

#include "AccountType.h"

namespace bank {

    class BankAccount {

    public:

        void populate(int number, double balance);

        [[nodiscard]] auto getNumber() const -> long;

        [[nodiscard]] auto getBalance() const -> double;

        [[nodiscard]] auto getType() const -> AccountType;

        auto setNumber(long number) -> void;

        auto setBalance(double balance) -> void;

        auto setType(AccountType type) -> void;

    private:

        long accNo = 0;

        double accBal = 0;

        AccountType accType = AccountType::Checking;

    };

}

```

## CreateAccount.cpp

```

//

// Created by dokto on 11.06.2024.

//

#include "CreateAccount.h"

#include "iostream"

#include "bank/AccountType.h"

```

```

/**
 * Executes the process of creating a bank account.
 *
 * This function creates a new bank account by calling the `creatBankAccount` function
 * and stores the result in the `bankAccount` variable. Then, it calls the `write`
 * function to display the details of the bank account.
 *
 * @return void
 */
auto bank::CreateAccount::process() -> void {
    const auto bankAccount = creatBankAccount();
    write(bankAccount);
}

/**
 * Writes the details of a bank account to the standard output.
 *
 * @param bankAccount the bank account to write
 *
 * @return void
 *
 * @throws None
 */
auto bank::CreateAccount::write(const bank::BankAccount &bankAccount) -> void {
    std::cout << "Account number is " << bankAccount.getNumber() << std::endl;
    std::cout << "Account balance is " << bankAccount.getBalance() << std::endl;
    std::cout << "Account type is " <<
        (bankAccount.getType() == bank::AccountType::Checking ? "Checking" : "Deposit") << std::endl;
}

```

```
}
```

```
/**
```

```
 * Creates a new bank account by prompting the user for account number and balance,
```

```
 * and sets the account type to Checking.
```

```
 *
```

```
 * @return A new BankAccount object with the entered account number, balance, and type.
```

```
 *
```

```
 * @throws None
```

```
 */
```

```
auto bank::CreateAccount::creatBankAccount() -> bank::BankAccount {
```

```
    bank::BankAccount bankAccount;
```

```
    std::cout << "Enter account number: ";
```

```
    long number;
```

```
    std::cin >> number;
```

```
    std::cout << "Enter the account balance! : ";
```

```
    double balance;
```

```
    std::cin >> balance;
```

```
    bankAccount.setNumber(number);
```

```
    bankAccount.setBalance(balance);
```

```
    bankAccount.setType(bank::AccountType::Checking);
```

```
    return bankAccount;
```

```
}
```

## **CreateAccount.h**

```
//
```

```
// Created by dokto on 11.06.2024.
```

```
//
```

```
#pragma once
```

```
#include "bank/bankAccount.h"
```

```
namespace bank {
```

```
    class CreateAccount {
```

```
    public:
```

```
        static auto process() -> void;
```

```
    private:
```

```
        static auto write(const bank::BankAccount &bankAccount) -> void;
```

```
        static auto creatBankAccount() -> bank::BankAccount;
```

```
};
```

```
}
```

## **main.cpp**

```
#include <bank/CreateAccount.h>
```

```
int main()
```

```
{
```

```
    bank::CreateAccount::process();
```

```
    return 0;
```

```
}
```

## **CmakeLists.txt**

```
cmake_minimum_required(VERSION 3.15.0)
```

```
include_guard(GLOBAL)
```

```
project(task10
```

```
    VERSION 0.0.1
```

```
    DESCRIPTION "task8 for OOP"
```

```
    LANGUAGES C CXX
```

```
)
```

```
if(NOT CMAKE_CXX_STANDARD)
```

```
    message(STATUS "[${PROJECT_NAME}] setting c++ standard to c++23")
```

```
    set(CMAKE_CXX_STANDARD 23)
```

```
    set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

```
    set(CMAKE_CXX_EXTENSIONS OFF)
```

```
endif()
```

```
add_executable(${PROJECT_NAME}
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/main.cpp
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/Bank/BankAccount.cpp
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/Bank/BankAccount.h
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/bank/AccountType.h
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/bank/CreateAccount.cpp
```

```
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++/bank/CreateAccount.h
)

target_include_directories(${PROJECT_NAME}
    PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/src/c++
)
```

## **5. Полученные результаты**

В ходе выполнения данной лабораторной работы нами были получены следующие результаты:

- в ходе работы программы были созданы методы и классы, с помощью которых создается два банковских аккаунта, в третьем упражнении с счета можно снимать деньги и класть их, также на экран выводится вся информация об аккаунтах.

## **6. Выводы**

В ходе выполнения данной лабораторной работы:

- были изучены классы в языке C++;