

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

ОТЧЕТ
по лабораторной работе №8
по дисциплине «Объектно-ориентированное
программирование»
Тема: «Применение конструкторов»

Студенты гр. 1335

_____ Максимов Ю.Е.

_____ Новакова Н. Е.

Преподаватель:

_____ Васильев А.А.

Санкт-Петербург

2024

1. Цель работы

Изучение конструкторов в языке C++ с помощью программного продукта компании CLion.

2. Анализ задачи

Необходимо:

- 1) Написать программу, которая изменяет программу, написанную в прошлой лабораторной, нужно добавить конструктор по умолчанию и еще три конструктора с различной комбинацией параметров;
- 2) Написать программу, которая добавляет к прошлой программе новый класс BankDeposit;
- 3) Написать программу, которая добавляет в предыдущую программу деструктор.

3. Ход выполнения работы

3.1 Упражнение 1

В ходе выполнения данного упражнения написана программа, которая преобразовывает программу из прошлой лабораторной работы, добавляя в нее четыре различных конструктора (один по умолчанию, три с различной комбинацией параметров).

3.1.1 Пошаговое описание алгоритма

Банковским аккаунтам присваиваются значения с помощью конструкторов.

С помощью методов Deposit, Withdraw изменяется баланс аккаунтов.

С помощью метода Write выводятся все значения 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;

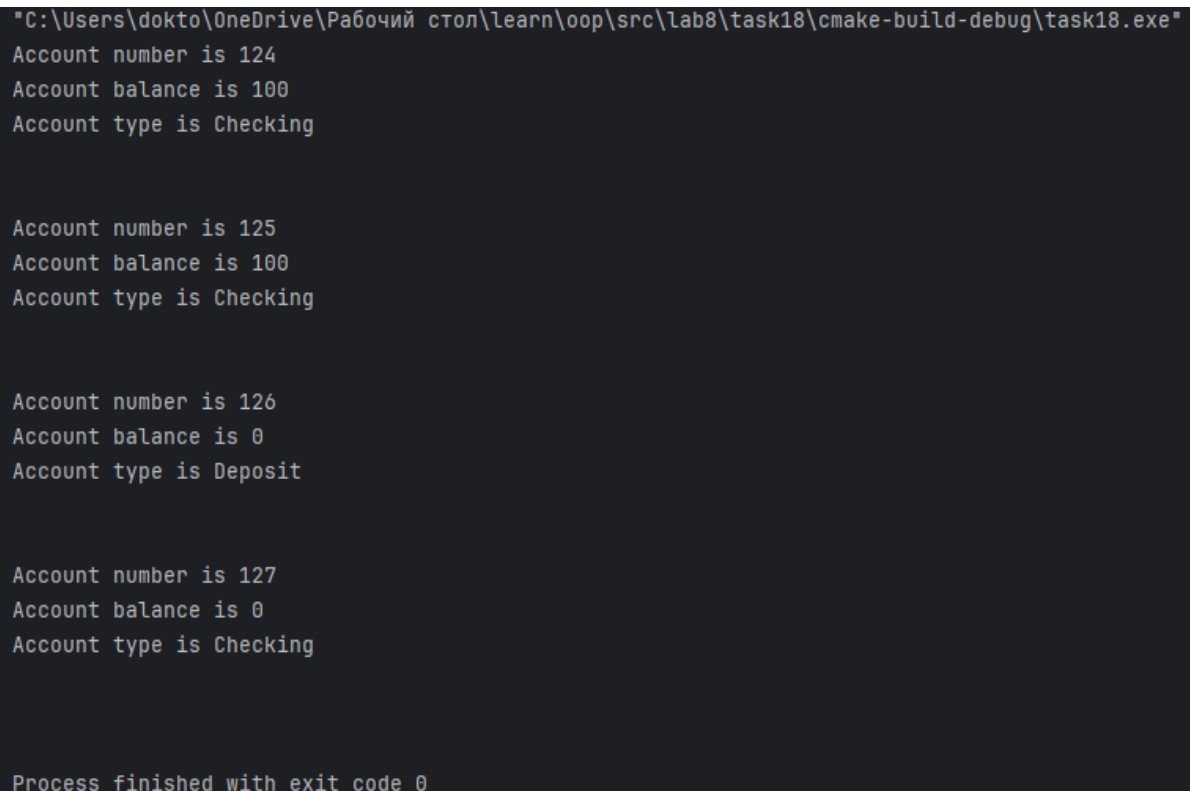
- `std::cin` – ожидает следующего нажатия клавиши пользователем;

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

- `BankAccount` – класс, который описывает банковский аккаунт с помощью методов: `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта), `Withdraw` (позволяет произвести снятие денег) и `Deposit` (позволяет пополнить счет), `Transaction` (возвращает информацию обо всех транзакциях, произведенных с данного аккаунта).

3.1.3 Контрольный пример

На рис.3.1 представлены результаты выполнения программы.



```
"C:\Users\dokto\OneDrive\Рабочий стол\learn\oop\src\lab8\task18\cmake-build-debug\task18.exe"
Account number is 124
Account balance is 100
Account type is Checking

Account number is 125
Account balance is 100
Account type is Checking

Account number is 126
Account balance is 0
Account type is Deposit

Account number is 127
Account balance is 0
Account type is Checking

Process finished with exit code 0
```

Рис.3.1 Контрольный пример для программы

Как видно из рисунка, на экран выводится вся информация об аккаунтах.

3.2 Упражнение 2

В ходе выполнения данного упражнения, написана программа, которая вносит изменения в прошлую программу, добавляя один класс, описывающий транзакцию.

3.2.1 Пошаговое описание алгоритма

Банковским аккаунтам присваиваются значения с помощью конструкторов.

С помощью методов Deposit, Withdraw изменяется баланс аккаунтов.

С помощью метода Write выводятся все значения аккаунтов.

3.2.2 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: `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта), `Withdraw` (позволяет произвести снятие денег) и `Deposit` (позволяет пополнить счет), `Transaction` (возвращает информацию обо всех транзакциях, произведенных с данного аккаунта).

- BankTransaction – класс, который описывает транзакцию, включая методы Amount и When, возвращающие количество переведенных денег и дату произведения операции.

3.2.3 Контрольный пример

На рис.3.2 представлены результаты выполнения программы.

```
"C:\Users\dokto\OneDrive\Рабочий стол\learn\oop\src\lab8\task18\cmake-build-debug\task18.exe"
Account number is 124
Account balance is 110
Account type is Checking
Transaction history:
10
13 June

Account number is 125
Account balance is 80
Account type is Checking
Transaction history:
20
13 June

Account number is 126
Account balance is 0
Account type is Deposit
Transaction history:

Account number is 127
Account balance is 0
Account type is Checking
Transaction history:

File has been written
20 13 June
File has been written
10 13 June

Process finished with exit code 0
```

Рис.3.2 Контрольный пример для программы

Как видно из рисунка, на экран выводятся вся информация об аккаунтах, дата и время.

3.3 Упражнение 3

В ходе выполнения данного упражнения, написана программа, которая дополняет прошлую программу, добавляя деструктор.

3.3.1 Пошаговое описание алгоритма

Банковским аккаунтам присваиваются значения с помощью конструкторов.

С помощью методов Deposit, Withdraw изменяется баланс аккаунтов.

С помощью метода Write выводятся все значения аккаунтов.

С помощью деструктора значения аккаунтов выводятся в файл.

3.3.2 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: `Number` (возвращает номер аккаунта), `Balance` (возвращает текущий баланс), `Type` (возвращает тип аккаунта), `Withdraw` (позволяет произвести снятие денег) и `Deposit` (позволяет пополнить счет), `Transaction` (возвращает информацию обо всех транзакциях, произведенных с данного аккаунта).

- `BankTransaction` – класс, который описывает транзакцию, включая методы `Amount` и `When`, возвращающие количество переведенных денег и дату произведения операции, и деструктор.

3.3.3 Контрольный пример

На рис.3.3 представлены результаты выполнения программы.

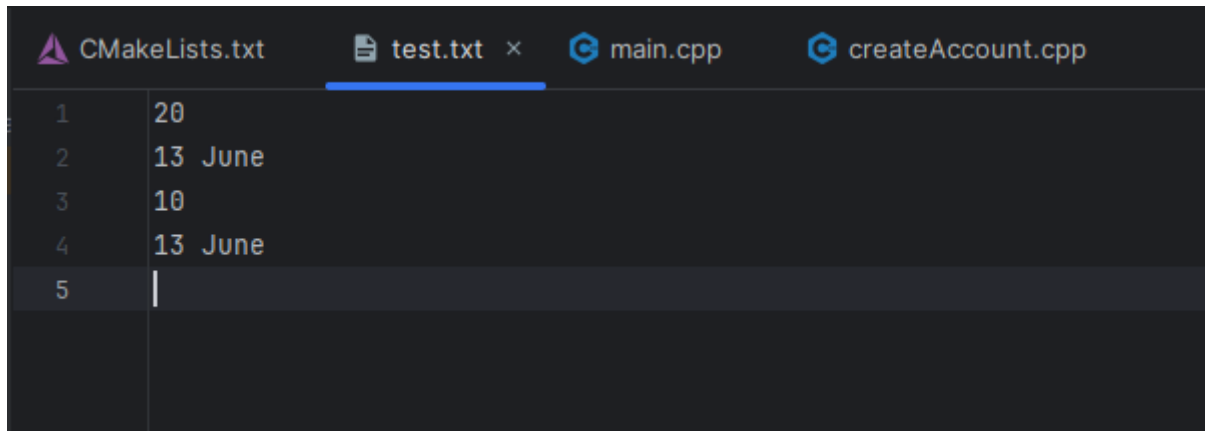
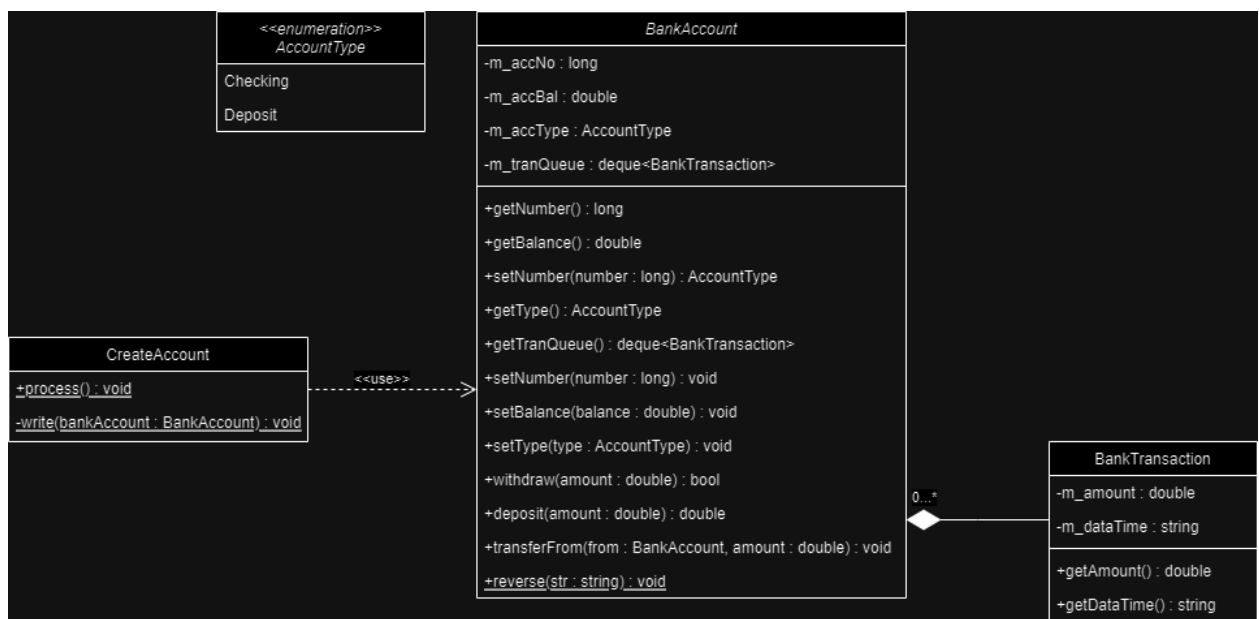


Рис.3.3 Контрольный пример для программы

Как видно из рисунка, на экран выводятся вся информация об аккаунтах, дата и время.

3.3.4 Диаграмма



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"
#include "algorithm"

```

```

namespace {
    long nextNumber = 123;

```

```

    /**
     * Increments and returns the next number in the sequence.
     *
     * @return The next number in the sequence.
     */

```

```

    auto NextNumber() -> long {
        return ++nextNumber;
    }

```

```

}
/**
 * Constructs a BankAccount object with default values.
 *
 * This constructor initializes the account number, balance, and type with default values.
 * The account number is generated using the `nextNumber()` function. The balance is set to 0.
 * The account type is set to `AccountType::Checking`.
 *
 * @return None.
 *
 * @throws None.
 */

```

```

bank::BankAccount::BankAccount():
    m_accNo(::NextNumber()),
    m_accBal(0),
    m_accType(AccountType::Checking)
{
}

```

```

/**
 * Constructs a BankAccount object with the given account type and initializes the account number,
 * balance, and type.
 *
 * @param type The account type of the BankAccount object.
 *
 * @return None.
 *
 * @throws None.
 */

```

```

bank::BankAccount::BankAccount(const AccountType type, const double balance):
    m_accNo(::NextNumber()),
    m_accBal(balance),
    m_accType(type)

```



```

{
}

bank::BankAccount::BankAccount(const AccountType type):
    m_accNo(::NextNumber()),
    m_accBal(0),
    m_accType(type)
{
}

/**
 * Constructs a BankAccount object with the given balance and sets the account number, balance, and type.
 *
 * @param balance The initial balance of the account.
 *
 * @throws None
 */
bank::BankAccount::BankAccount(const double balance):
    m_accNo(::NextNumber()),
    m_accBal(balance),
    m_accType(AccountType::Checking)
{
}

/**
 * Returns the account number.
 *
 * @return The account number.
 *
 * @throws None.
 */
auto bank::BankAccount::getNumber() const -> long {
    return m_accNo;
}

/**
 * Returns the account balance.
 *
 * @return The account balance.
 *
 * @throws None.
 */
auto bank::BankAccount::getBalance() const -> double {
    return m_accBal;
}

/**
 * Returns the account type.
 *
 * @return The account type.
 *
 * @throws None.
 */
auto bank::BankAccount::getType() const -> AccountType {
    return m_accType;
}

/**
 * Sets the account number.
 *

```

```

* @param number The account number.
*
* @throws None.
*/
auto bank::BankAccount::setNumber(const long number) -> void {
    m_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 {
    m_accBal = balance;
}

/**
* Sets the account type.
*
* @param type The new account type.
*
* @return void
*
* @throws None
*/
auto bank::BankAccount::setType(const AccountType type) -> void {
    m_accType = type;
}

/**
* Withdraws the specified amount from the bank account if there are sufficient funds.
*
* @param amount The amount to be withdrawn.
*
* @return True if the withdrawal was successful, false otherwise.
*
* @throws None
*/
auto bank::BankAccount::withdraw(const double amount) -> bool {
    const auto sufficientFunds = m_accBal >= amount;
    if (sufficientFunds) {
        m_tranQueue.emplace_back(amount);
        m_accBal -= amount;
    }
    return sufficientFunds ;
}

/**
* Deposits the specified amount into the bank account.
*
* @param amount The amount to be deposited.
*
* @return The updated balance of the bank account after the deposit.
*
* @throws None
*/
auto bank::BankAccount::deposit(const double amount) -> double {

```

```

        m_tranQueue.emplace_back(amount);
        return m_accBal += amount;
    }

/**
 * Transfers the specified amount from the given BankAccount to the current BankAccount.
 *
 * @param from The BankAccount to transfer the amount from.
 * @param amount The amount to be transferred.
 *
 * @return void
 *
 * @throws None
 */
auto bank::BankAccount::transferFrom(BankAccount &from, const double amount) -> void {
    if(from.withdraw(amount)) {
        this->deposit(amount);
    }
}

auto bank::BankAccount::getTranQueue() const -> const std::deque<BankTransaction>& {
    return m_tranQueue;
}

/**
 * Reverses the given string in-place.
 *
 * @param str The string to be reversed.
 *
 * @throws None
 */
void bank::BankAccount::reverse(std::string &str) {
    std::ranges::reverse(str);
}

```

bankAccount.h

```

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

#pragma once

#include "accountType.h"
#include "queue"
#include "bank/bankTransaction.h"

namespace bank {
    class BankAccount {

    public:
        explicit BankAccount();
        explicit BankAccount(AccountType type, double balance);
        explicit BankAccount(AccountType type);
        explicit BankAccount(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;
        auto withdraw(double amount) -> bool;
        auto deposit(double amount) -> double;
        auto transferFrom(BankAccount &from, double amount) -> void;
    };
}

```

```

        [[nodiscard]] auto getTranQueue() const -> const std::deque<BankTransaction>&;
        static auto reverse(std::string & str) -> void;
    private:
        long m_accNo;
        double m_accBal;
        AccountType m_accType;
        std::deque<BankTransaction> m_tranQueue;
    };
}

```

bankTransaction.cpp

```

//
// Created by Юлий Максимов on 13.06.2024.
//

#include "bankTransaction.h"

#include <ostream>
#include <fstream>
#include <iostream>

/**
 * Constructor for BankTransaction class.
 *
 * @param amount The amount for the transaction
 *
 */
bank::BankTransaction::BankTransaction(const double amount):
    m_amount(amount),
    m_dataTime("13 June"){
}

/**
 * Destructor for BankTransaction class.
 *
 * @param None
 *
 * @return None
 *
 * @throws std::logic_error if the file cannot be opened for writing
 */
bank::BankTransaction::~BankTransaction() {
    std::ofstream out; // поток для записи
    out.open("../test.txt", std::ios_base::app); // открываем файл для записи
    if (out.is_open())
    {
        std::cout << "File has been written" << std::endl << m_amount << " " << m_dataTime << std::endl;
        out << m_amount << std::endl;
        out << m_dataTime << std::endl;
    }else {
        throw std::logic_error("not open file");
    }
    out.close();
}

/**
 * Retrieves the amount associated with the BankTransaction object.
 *
 * @return The amount as a double.
 */
auto bank::BankTransaction::getAmount() const -> double {
    return m_amount;
}

```

```
}
```

```
/**
```

```
* Retrieves the date and time associated with the BankTransaction object.
```

```
*
```

```
* @return The date and time as a string.
```

```
*/
```

```
auto bank::BankTransaction::getDateTime() const -> std::string {  
    return m_dataTime;
```

```
}
```

bankTransaction.h

```
//
```

```
// Created by Юлий Максимов on 13.06.2024.
```

```
//
```

```
#pragma once
```

```
#include <string>
```

```
namespace bank {
```

```
    class BankTransaction {
```

```
    public:
```

```
        explicit BankTransaction(double amount);
```

```
        virtual ~BankTransaction();
```

```
        [[nodiscard]] auto getAmount() const -> double;
```

```
        [[nodiscard]] auto getDateTime() const -> std::string;
```

```
    private:
```

```
        double m_amount;
```

```
        std::string m_dataTime;
```

```
    };
```

```
}
```

createAccount.cpp

```
//
```

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

```
//
```

```
#include "createAccount.h"
```

```
#include "iostream"
```

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

```
/**
```

```
* Processes the creation of bank accounts, performs various operations like deposit, withdrawal, and transfers.
```

```
*
```

```
* @return None
```

```
*
```

```
* @throws None
```

```
*/
```

```
auto bank::CreateAccount::process() -> void {  
    auto bankAccount1 = BankAccount(bank::AccountType::Checking, 100);  
    auto bankAccount2 = BankAccount(100);  
    auto bankAccount3 = BankAccount(bank::AccountType::Deposit);  
    auto bankAccount4 = BankAccount();  
    bankAccount1.deposit(10);  
    bankAccount2.withdraw(20);  
    bankAccount3.transferFrom(bankAccount4,30);  
    write(bankAccount1);  
    write(bankAccount2);  
    write(bankAccount3);  
    write(bankAccount4);  
}
```

```

/**
 * 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;
    std::cout << "Transaction history: " << std::endl;
    for (const auto &tran : bankAccount.getTranQueue()) {
        std::cout << tran.getAmount() << std::endl;
        std::cout << tran.getDataTime();
    }
    std::cout << std::endl << std::endl;
}

```

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;
    };
}

```

Main.cpp

```

#include <iostream>
#include "utils/utils.h"
#include "cord/Cordinate.h"
#include "memory"

/**
 * The main function of the C++ program.
 *
 * @return 0 indicating successful execution.
 *
 * @throws None.
 */
int main()
{
    const std::unique_ptr<ICordinate> cord = std::make_unique<Cordinate>(21.0,68.0);
    utils::Utils::display(cord.get());
    return 0;
}

```

5. Полученные результаты

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

- В ходе работы программы были созданы методы, конструкторы, деструкторы и классы для работы с банковскими аккаунтами.

6. Выводы

В ходе выполнения данной лабораторной работы:

- были изучены конструкторы и деструкторы в языке C++;