

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 5
з дисципліни «Основи програмування – 2.
Методології програмування»

«Успадкування та поліморфізм»

Варіант 24

Виконав студент ПІ-13 Нещерет Віталій Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірів Вечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Лабораторна робота 5

Успадкування та поліморфізм

Варіант 24

24. Створити клас «Банківський рахунок», який містить назву банку, номер рахунку, статус (діє, закінчився) та методи додавання / зняття коштів. На основі цього класу створити класи-нащадки «Депозитний рахунок», який додатково містить дату його відкриття, період, ставку, суму коштів, та «Розрахунковий рахунок», який додатково містить дату останньої операції, залишок коштів на рахунку. Створити n пар депозитних та розрахункових рахунків. В межах вказаного періоду передбачити щомісячне перерахування процентів по депозиту на відповідний розрахунковий рахунок. У випадку закінчення терміну дії депозиту, перерахувати на відповідний розрахунковий рахунок усі кошти (разом з процентами) та закрити відповідний депозитний рахунок.

Код програми

C++

lab_cpp_5.cpp

```
#include "Header.h"

int main()
{
    cout << "Enter the number of bank accounts: ";
    int n;
    cin >> n;
    cin.ignore();

    vector<CurrentAccount> current;
    vector<DepositAccount> deposits;
    initVectors(n, current, deposits);
    checkDeposits(n, current, deposits);
    printAccounts(n, current, deposits);

    return 0;
}
```

Header.h

```
#pragma once
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Date {
private:
    int year;
    int month;
```

```

        int day;
public:
    void setYear(int yr) { year = yr; };
    void setMonth(int mon) { month = mon; };
    void setDay(int d) { day = d; };
    int getYear() { return year; };
    int getMonth() { return month; };
    int getDay() { return day; };
    string getString();
    Date(int day = 1, int month = 1, int year = 1900);
    Date(string);
    friend int getMonthsBetweenDates(Date&, Date&);
};

class BankAccount {
private:
    string bankName;
    int id;
    bool isAvailable;
public:
    BankAccount(vector<string>);
    void setIsAvailable(bool isAvailable) { this->isAvailable = isAvailable; };
    int getId() { return id; };
    bool getIsAvailable() { return isAvailable; };
    string getBankName() { return bankName; };
    virtual void addBalance(double) = 0;
    virtual void subBalance(double) = 0;
};

class CurrentAccount : public BankAccount {
private:
    Date lastOperationDate;
    double balance;
public:
    CurrentAccount(vector<string>);
    void setLastOperationDate(Date date) { lastOperationDate = date; };
    double getBalance() { return balance; };
    Date getLastOperationDate() { return lastOperationDate; };
    virtual void addBalance(double sum) { balance += sum; };
    virtual void subBalance(double sum) { balance -= sum; };
};

class DepositAccount : public BankAccount {
private:
    Date openingDate;
    int period;
    double balance;
    double rate;
public:
    DepositAccount(vector<string>);
    Date getOpeningDate() { return openingDate; };
    int getPeriod() { return period; };
    double getBalance() { return balance; };
    double getRate() { return rate; };
    virtual void addBalance(double sum) { balance += sum; };
    virtual void subBalance(double sum) { balance -= sum; };
};

void initVectors(int, vector<CurrentAccount>&, vector<DepositAccount>&);
void checkDeposits(int, vector<CurrentAccount>&, vector<DepositAccount>&);
void printAccounts(int, vector<CurrentAccount>&, vector<DepositAccount>&);
Date getSystemDate();
vector<string> split(string, char);

```

Source.cpp

```
#include "Header.h"

BankAccount::BankAccount(vector<string> attributes) {
    bankName = attributes[0];
    id = stoi(attributes[1]);
    isAvailable = true;
}

CurrentAccount::CurrentAccount(vector<string> attributes) : BankAccount(attributes) {
    balance = stod(attributes[2]);
    lastOperationDate = Date();
}

DepositAccount::DepositAccount(vector<string> attributes) : BankAccount(attributes) {
    balance = stod(attributes[2]);
    rate = stod(attributes[3]);
    openingDate = Date(attributes[4]);
    period = stoi(attributes[5]);
}

Date::Date(string str) {
    vector<string> temp = split(str, '.');
    day = stoi(temp[0]);
    month = stoi(temp[1]);
    year = stoi(temp[2]);
}

Date::Date(int day, int month, int year) {
    this->year = year;
    this->month = month;
    this->day = day;
}

string Date::getString() {
    string strDay = to_string(day);
    string strMonth = to_string(month);
    if (day < 10) {
        strDay = '0' + to_string(day);
    }
    if (month < 10) {
        strMonth = '0' + to_string(month);
    }
    return strDay + '.' + strMonth + '.' + to_string(year);
}

int getMonthsBetweenDates(Date& date1, Date& date2) {
    return ((date1.year - date2.year) * 12) + date1.month - date2.month;
}

void initVectors(int size, vector<CurrentAccount>& currents, vector<DepositAccount>&
deposits) {
    cout << "Enter info about accounts in format" << endl;
    cout << "For current account: [bank name, id, balance]" << endl;
    cout << "For deposit account: [bank name, id, balance, rate, dd.mm.yyyy, period]"
<< endl;
    cout << "and again..." << endl;

    for (int i = 0; i < size; i++)
    {
```

```

        string str;
        vector<string> attributes;
        getline(cin, str);
        attributes = split(str, ',');
        currents.push_back(CurrentAccount(attributes));
        getline(cin, str);
        attributes = split(str, ',');
        deposits.push_back(DepositAccount(attributes));
        cout << endl;
    }
}

void checkDeposits(int size, vector<CurrentAccount>& currents, vector<DepositAccount>&
deposits) {
    Date now = getSystemDate();
    for (int i = 0; i < size; i++)
    {
        Date openDate = deposits[i].getOpeningDate();
        int completedMonth = getMonthsBetweenDates(now, openDate);
        double percents;

        if (completedMonth >= deposits[i].getPeriod())
        {
            percents = deposits[i].getPeriod() * (deposits[i].getBalance() *
deposits[i].getRate());
            currents[i].addBalance(deposits[i].getBalance());
            deposits[i].subBalance(deposits[i].getBalance());
            deposits[i].setIsAvailable(false);
            openDate.setMonth(openDate.getMonth() + deposits[i].getPeriod());
        }
        else
        {
            percents = completedMonth * (deposits[i].getBalance() *
deposits[i].getRate());
            openDate.setMonth(openDate.getMonth() + completedMonth);
        }
        while (openDate.getMonth() > 12)
        {
            openDate.setYear(openDate.getYear() + 1);
            openDate.setMonth(openDate.getMonth() - 12);
        }
        currents[i].addBalance(percents);
        currents[i].setLastOperationDate(openDate);
    }
}

void printAccounts(int size, vector<CurrentAccount>& currents, vector<DepositAccount>&
deposits) {
    cout << endl;
    for (int i = 0; i < size; i++)
    {
        CurrentAccount c = currents[i];
        DepositAccount d = deposits[i];
        Date lastOp = c.getLastOperationDate();
        string status = d.getIsAvailable() ? "Available" : "Unavailable";

        cout << "Pair " << i + 1 << ": " << endl;

        cout << "id:" << c.getId()
            << "\tbank name: " << c.getBankName()
            << "\tdate of last operation: " << lastOp.getString()
            << "\tbalance: " << c.getBalance() << endl;

        cout << "id:" << d.getId()
            << "\tbank name: " << d.getBankName()

```

```

        << "\tstatus: " << status
        << "\tbalance: " << d.getBalance() << endl << endl;
    }
}

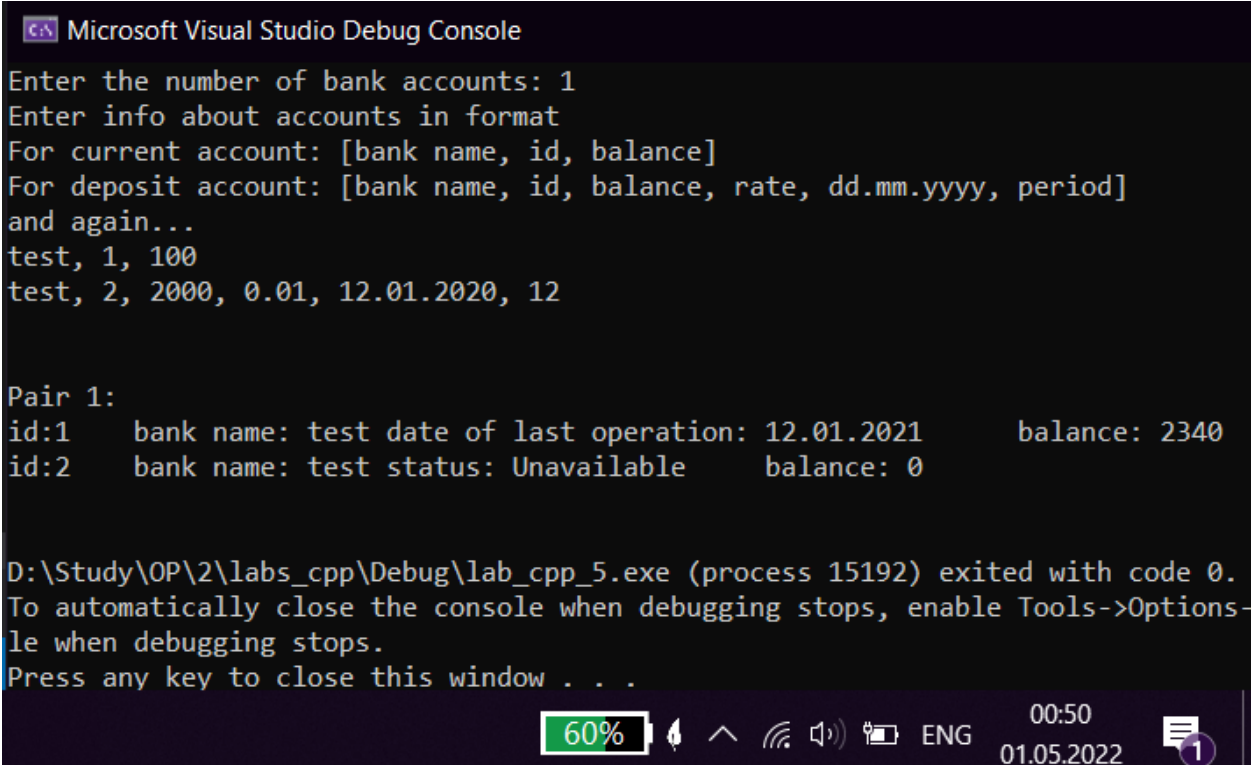
Date getSystemDate()
{
    struct tm currentTime;
    time_t t = time(NULL);
    localtime_s(&currentTime, &t);

    return Date(currentTime.tm_mday, currentTime.tm_mon + 1,
currentTime.tm_year + 1900);
}

vector<string> split(string str, char separator) {
    vector<string> res;
    string slice = "";
    str += " ";
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == separator)
        {
            if (slice.length() > 0) {
                res.push_back(slice);
                slice = "";
            }
        }
        else {
            slice += str[i];
        }
    }
    res.push_back(slice);
    return res;
}

```

Тестування:



```

Microsoft Visual Studio Debug Console

Enter the number of bank accounts: 1
Enter info about accounts in format
For current account: [bank name, id, balance]
For deposit account: [bank name, id, balance, rate, dd.mm.yyyy, period]
and again...
test, 1, 100
test, 2, 2000, 0.01, 12.01.2020, 12

Pair 1:
id:1    bank name: test date of last operation: 12.01.2021    balance: 2340
id:2    bank name: test status: Unavailable    balance: 0

D:\Study\OP\2\labs_cpp\Debug\lab_cpp_5.exe (process 15192) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
le when debugging stops.
Press any key to close this window . . .

60% 00:50 01.05.2022 1

```

Python

lab_py_5.py

```
from BankAccount import *

n = int(input("Enter the number of bank accounts: "))
currents = []
deposits = []
init_arrays(n, currents, deposits)
check_deposits(n, currents, deposits)
print_accounts(n, currents, deposits)
```

BankAccount.py

```
from abc import ABC, abstractmethod
from datetime import datetime

class Date:
    __year = 1900
    __month = 1
    __day = 1

    def __init__(self, day=1, month=1, year=1900):
        self.__day = day
        self.__month = month
        self.__year = year

    def set_year(self, year):
        self.__year = year

    def set_month(self, month):
        self.__month = month

    def set_day(self, day):
        self.__day = day

    def get_year(self):
        return self.__year

    def get_month(self):
        return self.__month

    def get_day(self):
        return self.__day

    def __str__(self):
        return "{}.{}.{}".format(self.__year, self.__month, self.__day)

    def get_month_between_dates(self, other):
        return ((self.__year - other.get_year()) * 12) + self.__month -
other.get_month()

class BankAccount(ABC):
    __bank_name = ""
    __id = 0
    __is_available = True

    def __init__(self, attributes):
```

```

        self.__bank_name = attributes[0]
        self.__id = int(attributes[1])
        self.__is_available = True

    def set_is_available(self, status):
        self.__is_available = status

    def get_bank_name(self):
        return self.__bank_name

    def get_id(self):
        return self.__id

    def get_is_available(self):
        return self.__is_available

    @abstractmethod
    def add_balance(self, a):
        pass

    @abstractmethod
    def sub_balance(self, a):
        pass

class CurrentAccount(BankAccount):
    __last_operation_date = Date()
    __balance = 0.0

    def __init__(self, attributes):
        super(CurrentAccount, self).__init__(attributes)
        self.__balance = float(attributes[2])
        self.__last_operation_date = Date()

    def set_last_operation_date(self, date):
        self.__last_operation_date = date

    def get_last_operation_date(self):
        return self.__last_operation_date

    def get_balance(self):
        return self.__balance

    def add_balance(self, a):
        self.__balance += a

    def sub_balance(self, a):
        self.__balance -= a

class DepositAccount(BankAccount):
    __opening_date = Date()
    __period = 0
    __balance = 0.0
    __rate = 0.0

    def __init__(self, attributes):
        super(DepositAccount, self).__init__(attributes)
        self.__balance = float(attributes[2])
        self.__rate = float(attributes[3])
        att_for_date = attributes[4].split('.')
        self.__opening_date = Date(int(att_for_date[0]),
int(att_for_date[1]), int(att_for_date[2]))
        self.__period = int(attributes[5])

```



```

def get_opening_date(self):
    return self.__opening_date

def get_period(self):
    return self.__period

def get_balance(self):
    return self.__balance

def get_rate(self):
    return self.__rate

def add_balance(self, a):
    self.__balance += a

def sub_balance(self, a):
    self.__balance -= a

def init_arrays(size, currents, deposits):
    print("Enter info about accounts in format\n"
          "For current account: [bank name, id, balance]\n"
          "For deposit account: [bank name, id, balance, rate, dd.mm.yyyy,"
          period]\n"
          "and again...")
    for i in range(size):
        attributes = input().split(',')
        currents.append(CurrentAccount(attributes))
        attributes = input().split(',')
        deposits.append(DepositAccount(attributes))
        print()

    return currents, deposits

def check_deposits(size, currents, deposits):
    now = get_system_date()
    for i in range(size):
        open_date = deposits[i].get_opening_date()
        completed_month = now.get_month_between_dates(open_date)
        if completed_month >= deposits[i].get_period():
            percents = deposits[i].get_period() * (deposits[i].get_balance()
            * deposits[i].get_rate())
            currents[i].add_balance(deposits[i].get_balance())
            deposits[i].sub_balance(deposits[i].get_balance())
            deposits[i].set_is_available(False)
            open_date.set_month(open_date.get_month() +
            deposits[i].get_period())
        else:
            percents = completed_month * (deposits[i].get_balance() *
            deposits[i].get_rate())
            open_date.set_month(open_date.get_month() + completed_month)
            while open_date.get_month() > 12:
                open_date.set_year(open_date.get_year() + 1)
                open_date.set_month(open_date.get_month() - 12)
            currents[i].add_balance(percents)
            currents[i].set_last_operation_date(open_date)

def print_accounts(size, currents, deposits):
    print()
    for i in range(size):
        c = currents[i]

```

```

        d = deposits[i]
        last_op = c.get_last_operation_date()
        status = "Available" if d.get_is_available() else "Unavailable"
        print("Pair {}:\n"
              "id: {}"
              "\tbank name: {}"
              "\tdate of last operation: {}"
              "\tbalance: {}"
              "\nid: {}"
              "\tbank name: {}"
              "\tstatus: {}"
              "\tbalance: {}\n"
              .format(str(i + 1), c.get_id(), c.get_bank_name(),
last_op.__str__(), c.get_balance(),
                      d.get_id(), d.get_bank_name(), status,
d.get_balance()))

def get_system_date():
    current_datetime = datetime.now()
    sys_date = Date(current_datetime.day, current_datetime.month,
current_datetime.year)
    return sys_date

```

Тестування:

```

lab_py_5 x
D:\Study\Code\2\Scripts\python.exe D:/Study/OP/2/labs_py/lab_py_5/lab_py_5.py
Enter the number of bank accounts: 1
Enter info about accounts in format
For current account: [bank name, id, balance]
For deposit account: [bank name, id, balance, rate, dd.mm.yyyy, period]
and again...
test, 123, 113
test, 321, 1488, 0.13, 01.01.2022, 12

Pair 1:
id: 123 bank name: test date of last operation: 05.5.2022   balance: 886.76
id: 321 bank name: test status: Available   balance: 1488.0

Process finished with exit code 0

```

Висновки:

Я вивчив механізми успадкування класів та поліморфізму. Застосував ці навички на практиці.