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. Модульне програмування»

«Успадкування та поліморфізм»

Варіант 24

Виконав студент ІП-11 Печковський Олександр Костянтинович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

(прізвище, ім'я, по батькові)

Київ 2022

**Умова задачі:**

Створити клас "Банківський рахунок", який містить назву банку, номер рахунку, статус (діє, закінчився) та методи додавання / зняття коштiв. На основі цього класу створити класи-нащадки "Депозитний рахунок", який додатково мiстить 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 вiдповiдний розрахунковий рахунок усі кошти (разом з процентами) та закрити відповідний депозитний рахунок.

**Код на С++:**

**Lab5C++.cpp**

#include "Header.h"

int main()

{

cout << "Enter the number of pairs of bank accounts: ";

int n;

cin >> n;

cin.ignore();

CurrentAccount \*\*currents = new CurrentAccount\*[n];

DepositAccount \*\*deposits = new DepositAccount\*[n];

CreatePairOfAccounts(n, currents, deposits);

BankOperations(n, currents, deposits);

AccountsOutput(n, currents, 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);

void setMonth(int);

void setDay(int);

int getYear();

int getMonth();

int getDay();

Date(string);

Date(int, int, int);

string getString();

friend int getMonthsBetweenDates(Date&, Date&);

};

class BankAccount {

private:

string bankName;

int account\_number;

bool isAvailable;

public:

BankAccount(vector<string>);

void setIsAvailable(bool);

int getAccountNumber();

bool getIsAvailable();

string getBankName();

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);

double getBalance();

Date getLastOperationDate();

void addBalance(double);

void subBalance(double);

};

class DepositAccount : public BankAccount {

private:

Date openingDate;

int period;

double balance;

double rate;

public:

DepositAccount(vector<string>);

Date getOpeningDate();

int getPeriod();

double getBalance();

double getRate();

void addBalance(double);

void subBalance(double);

};

void CreatePairOfAccounts(int, CurrentAccount\*\*&, DepositAccount\*\*&);

void BankOperations(int, CurrentAccount\*\*&, DepositAccount\*\*&);

void AccountsOutput(int, CurrentAccount\*\*&, DepositAccount\*\*&);

Date getSystemDate();

vector<string> split(string, char);

**Source.cpp**

#include "Header.h"

void Date::setYear(int yr) { year = yr; };

void Date::setMonth(int mon) { month = mon; };

void Date::setDay(int d) { day = d; };

int Date::getYear() { return year; };

int Date::getMonth() { return month; };

int Date::getDay() { return day; };

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 = 1, int month = 1, int year = 1900) {

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;

}

BankAccount::BankAccount(vector<string> attributes) {

bankName = attributes[0];

account\_number = stoi(attributes[1]);

isAvailable = true;

}

void BankAccount::setIsAvailable(bool isAvailable) { this->isAvailable = isAvailable; };

int BankAccount::getAccountNumber() { return account\_number; };

bool BankAccount::getIsAvailable() { return isAvailable; };

string BankAccount::getBankName() { return bankName; };

CurrentAccount::CurrentAccount(vector<string> attributes) : BankAccount(attributes) {

balance = stod(attributes[2]);

lastOperationDate = Date();

}

void CurrentAccount::setLastOperationDate(Date date) { lastOperationDate = date; };

double CurrentAccount::getBalance() { return balance; };

Date CurrentAccount::getLastOperationDate() { return lastOperationDate; };

void CurrentAccount::addBalance(double sum) { balance += sum; };

void CurrentAccount::subBalance(double sum) { balance -= sum; };

DepositAccount::DepositAccount(vector<string> attributes) : BankAccount(attributes) {

balance = stod(attributes[2]);

rate = stod(attributes[3]);

openingDate = Date(attributes[4]);

period = stoi(attributes[5]);

}

Date DepositAccount::getOpeningDate() { return openingDate; };

int DepositAccount::getPeriod() { return period; };

double DepositAccount::getBalance() { return balance; };

double DepositAccount::getRate() { return rate; };

void DepositAccount::addBalance(double sum) { balance += sum; };

void DepositAccount::subBalance(double sum) { balance -= sum; };

void CreatePairOfAccounts(int size, CurrentAccount\*\*& currents, DepositAccount\*\*& deposits) {

cout << "\nEnter info about accounts in following format:\n\n";

cout << "1st line - current account: [bank name, account number, balance]\n";

cout << "2nd line - deposit account: [bank name, account number, balance, monthly rate, start date, period]\n\n";

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

{

cout << "Enter pair " << i + 1<<":\n";

string str;

vector<string> attributes;

getline(cin, str);

attributes = split(str, ',');

currents[i]=new CurrentAccount(attributes);

getline(cin, str);

attributes = split(str, ',');

deposits[i]=new DepositAccount(attributes);

cout << endl;

}

}

void BankOperations(int size, CurrentAccount\*\*& currents, 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())/100);

(\*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())/100);

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 AccountsOutput(int size, CurrentAccount\*\*& currents, 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() ? "open" : "closed";

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

cout << "Bank name: " << c.getBankName()

<< "\nAccount number: " << c.getAccountNumber()

<< "\nDate of last operation: " << lastOp.getString()

<< "\nBalance: " << c.getBalance() << endl;

cout << "\nBank name: " << d.getBankName()

<< "\nAccount number: " << d.getAccountNumber()

<< "\nStatus: " << status

<< "\nBalance: " << 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];

}

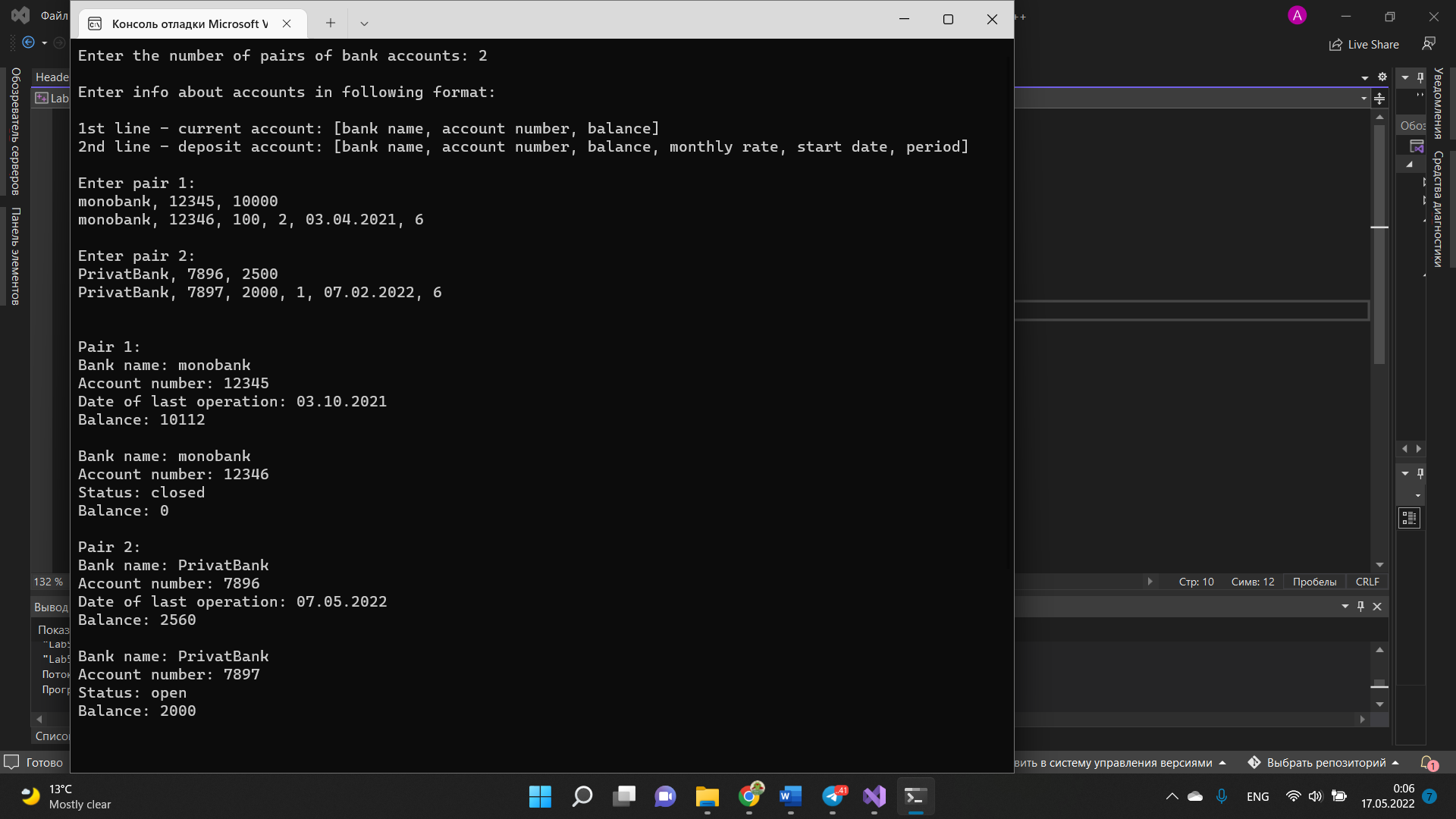
}

res.push\_back(slice);

return res;

}

**Скріншот роботи програми на C++:**

****

**Код на Python:**

**Lab5Python.py**

from BankAccount import \*

n = int(input("Enter the number of pairs of bank accounts: "))

currents\_array, deposits\_array=CreatePairOfAccounts(n)

BankOperations(n, currents\_array, deposits\_array)

AccountsOutput(n, currents\_array, deposits\_array)

**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):

d\_str = str(self.\_\_day)

if self.\_\_day < 10:

d\_str = '0' + str(self.\_\_day)

m\_str = str(self.\_\_month)

if self.\_\_month < 10:

m\_str = '0' + str(self.\_\_month)

return "{}.{}.{}".format(d\_str, m\_str, self.\_\_year)

def get\_month\_between\_dates(self, other):

return ((self.\_\_year - other.get\_year()) \* 12) + self.\_\_month - other.get\_month()

def get\_system\_date():

current\_datetime = datetime.now()

sys\_date = Date(current\_datetime.day, current\_datetime.month, current\_datetime.year)

return sys\_date

class BankAccount(ABC):

\_\_bank\_name = ""

\_\_account\_number = 0

\_\_is\_available = True

def \_\_init\_\_(self, attributes):

self.\_\_bank\_name = attributes[0]

self.\_\_account\_number = 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\_account\_number(self):

return self.\_\_account\_number

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 CreatePairOfAccounts(size):

currents\_array = []

deposits\_array = []

print("\nEnter info about accounts in following format:\n\n"

"1st line - current account: [bank name, account number, balance]\n"

"2nd line - deposit account: [bank name, account number, balance, monthly rate, start date, period]\n")

for i in range(size):

print("Enter pair "+str(i+1)+": ")

attributes = input().split(',')

currents\_array.append(CurrentAccount(attributes))

attributes = input().split(',')

deposits\_array.append(DepositAccount(attributes))

print()

return currents\_array, deposits\_array

def BankOperations(size, currents\_array, deposits\_array):

now = get\_system\_date()

for i in range(size):

open\_date = deposits\_array[i].get\_opening\_date()

completed\_month = now.get\_month\_between\_dates(open\_date)

if completed\_month >= deposits\_array[i].get\_period():

percents = deposits\_array[i].get\_period() \* (deposits\_array[i].get\_balance() \* (deposits\_array[i].get\_rate())/100)

currents\_array[i].add\_balance(deposits\_array[i].get\_balance())

deposits\_array[i].sub\_balance(deposits\_array[i].get\_balance())

deposits\_array[i].set\_is\_available(False)

open\_date.set\_month(open\_date.get\_month() + deposits\_array[i].get\_period())

else:

percents = completed\_month \* (deposits\_array[i].get\_balance() \* (deposits\_array[i].get\_rate())/100)

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\_array[i].add\_balance(percents)

currents\_array[i].set\_last\_operation\_date(open\_date)

def AccountsOutput(size, currents\_array, deposits\_array):

print()

for i in range(size):

c = currents\_array[i]

d = deposits\_array[i]

last\_op = c.get\_last\_operation\_date()

status = "open" if d.get\_is\_available() else "closed"

print("Pair {}:\n"

"Bank name: {}"

"\nAccount number: {}"

"\nDate of last operation: {}"

"\nBalance: {}"

"\n\nBank name: {}"

"\nAccount number: {}"

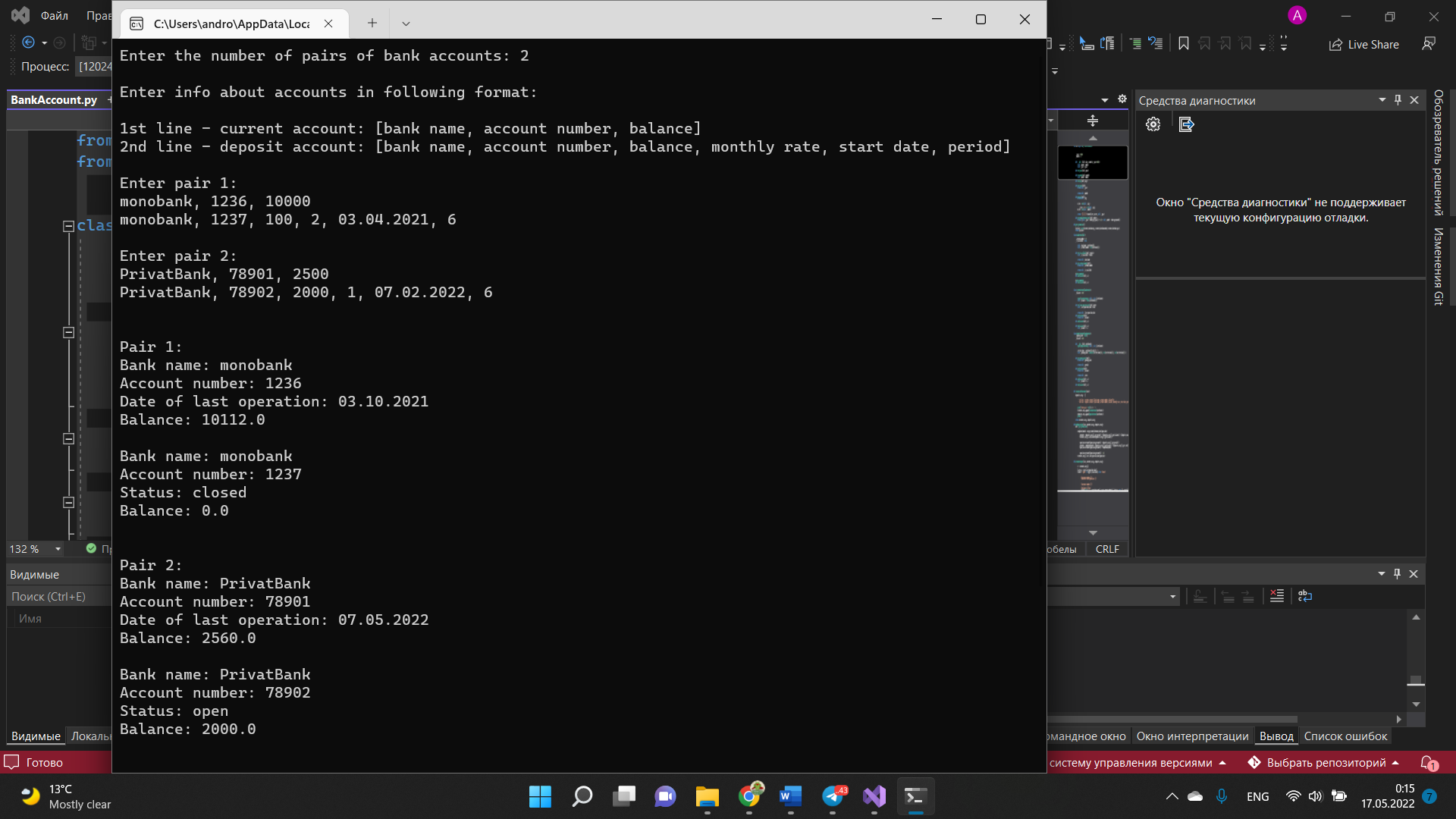
"\nStatus: {}"

"\nBalance: {}\n\n"

.format(str(i + 1), c.get\_bank\_name(), c.get\_account\_number(), last\_op.\_\_str\_\_(), c.get\_balance(),

d.get\_bank\_name(), d.get\_account\_number(), status, d.get\_balance()))

**Скріншот роботи програми на Python:**

****

**Висновок:** під час виконання лабораторної роботи я вивчив механізми успадкування та поліморфізму.