МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**«Разработка объектно-ориентированного приложения»**

Отчет по лабораторной работе № 6 по дисциплине

«Технологии программирования»

Выполнил студент группы: ПИб-2301\_\_\_\_\_\_\_\_\_\_\_\_\_ /Морозов Д.А.

Проверил преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л.

**Киров 2023**

**Цели работы:**

Выбрать предметную область.  Создать иерархию классов, состоящую не менее чем из одного родительского и двух дочерних классов. В каждом классе определить не менее  двух член данных. не менее двух собственных, а для дочерних не менее двух унаследованных и двух перекрытых член функций.  
Разработать приложение демонстрирующее принципы инкапсуляции, наследования и полиморфизма .  
Отчет должен содержать описание предметной области, схему структуры иерархии классов (можно использовать диаграмму классов), особенности реализации.   
\* Работу с объектами осуществлять только через переменные  родительского типа.  
Сделать выводы о проделанной работе.

**Предметная область:**

Структура классов представляет собой банковский аккаунт, родительский класс – «Bank» выполняет роль содержания основного баланса и номера лицевого счета с возможностью депозита/вывода средств.

Дочерний класс «SavingAccount» представляет из себя сберегательный счет, который имеет возможность использовать баланс для депозита на свой счет, начисление процентов на данный счет и т.д.

«CreditAccount» - кредитный счет, имеет методы, позволяющие осуществить взятие в долг денежных средств на основной баланс.

Методы для вывода всех полей класса и изменения одного поля класса, депозит/вывод средств являются переопределенными, а методы расчета процентных начислений, перевода валют - унаследованными.

**Диаграмма классов:**

Данная диаграмма классов отображает инкапсуляцию, наследование и полиморфизм.

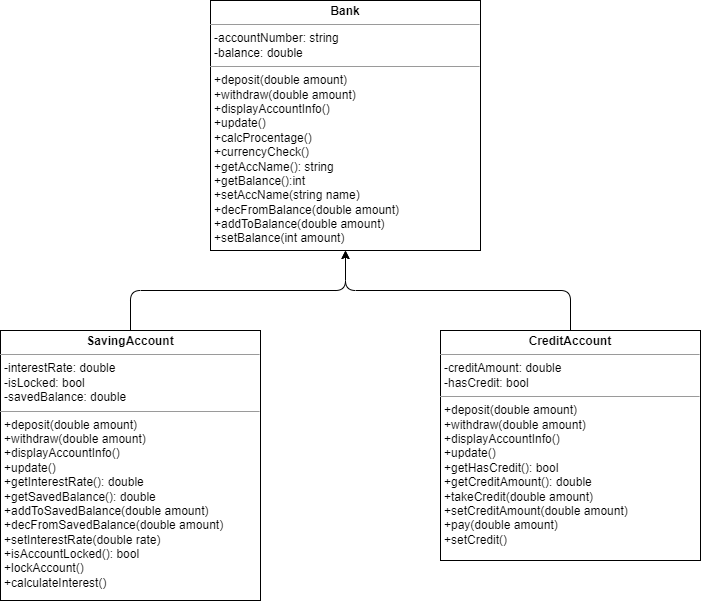


Рис. 1 – диаграмма классов

**Листинг кода:**

Bank.h

//

// Bank.h

// CPP\_Lab6

//

// Created by Danil Morozov on 16.05.2023.

//

#ifndef Bank\_h

#define Bank\_h

#include "helper.h"

using namespace std;

class BankAccount {

private:

string accountNumber;

double balance;

public:

BankAccount(string accountNumber, double balance) {

this -> accountNumber = accountNumber;

this -> balance = balance;

}

// Собственные методы

string getAccName() {

return this -> accountNumber;

}

int getBalance() {

return this -> balance;

}

void setAccName(string name) {

this -> accountNumber = name;

}

void decFromBalance(double amount) {

this -> balance -= amount;

}

void addToBalance(double amount) {

this -> balance -= amount;

}

void setBalance(int amount) {

this -> balance = amount;

}

// Методы которые будут переопределены

virtual void displayAccountInfo() {

cout << "Номер счета: " << getAccName() << endl;

cout << "Баланс: " << to\_string(getBalance()) << endl;

}

virtual void deposit(double amount) {

this -> balance += amount;

}

virtual void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

} else {

cout << "На счету недостаточно средств. Операция отклонена!" << endl;

};

}

virtual void update() {

string newValue;

int state;

cout << "1: Изменить номер счета" << endl;

cout << "2: Изменить баланс" << endl;

cout << "3: Внести депозит" << endl;

cout << "4: Вывести средства" << endl;

state = input();

cout << "Введите новое значение: ";

cin >> newValue;

switch (state)

{

case 1:

setAccName(newValue);

break;

case 2:

if (check(newValue)) {

setBalance(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 3:

if (check(newValue)) {

deposit(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 4:

if (check(newValue)) {

withdraw(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

default:

cout<< "Ошибка при выполнении операции!" << endl << endl;

break;

}

}

// Методы которые будут унаследованы

void calcProcentage(){

cout << "Введите предполагаемый годовой процентной ставки по вкладу: ";

double prc = input();

cout << "Выберете кол-во месяцев" << endl;

cout << "1. 6 месяцев" << endl;

cout << "2. 12 месяцев" << endl;

cout << "3. 24 месяца" << endl;

double resultProc = 0.0;

int months = input();

switch (months) {

case 1:

resultProc = (getBalance() \* prc \* 6 \* 30) / (365 \* 100);

break;

case 2:

resultProc = (getBalance() \* prc \* 6 \* 30) / (365 \* 100);

break;

case 3:

resultProc = (getBalance() \* prc \* 6 \* 30) / (365 \* 100);

break;

default:

cout << "Неверный выбор количества месяцев." << endl;

break;

}

cout << "Предполагаемые проценты по вкладу: " << resultProc << endl << endl;

}

void currencyCheck(){

double eurRate = 0.85; // Курс обмена Евро к Доллару США

double usdRate = 1.18; // Курс обмена Доллара США к Евро

double cnyRate = 6.47; // Курс обмена Юаня к Евро

double gbpRate = 0.71; // Курс обмена Фунта к Евро

double rubRate = 86.36; // Курс обмена Рубля к Евро

cout << "Введите сумму для перевода валюты: ";

int amount = input();

cout << "Выберете исходную валюту для перевода" << endl;

cout << "1. Евро" << endl;

cout << "2. Доллар" << endl;

cout << "3. Юань" << endl;

cout << "4. Фунт" << endl;

cout << "5. Рубль" << endl;

int newCurrency = input();

cout << "Выберете финальную валюту для перевода" << endl;

cout << "1. Евро" << endl;

cout << "2. Доллар" << endl;

cout << "3. Юань" << endl;

cout << "4. Фунт" << endl;

cout << "5. Рубль" << endl;

int oldCurrency = input();

double conversionRate = 0.0;

// Установка курса конверсии на основе выбранных валют

switch (oldCurrency) {

case 1: // Исходная валюта: Евро

switch (newCurrency) {

case 1:

conversionRate = 1.0;

break;

case 2:

conversionRate = eurRate / usdRate;

break;

case 3:

conversionRate = eurRate / cnyRate;

break;

case 4:

conversionRate = eurRate / gbpRate;

break;

case 5:

conversionRate = eurRate / rubRate;

break;

default:

cout << "Неверный выбор финальной валюты" << endl;

break;

}

break;

case 2: // Исходная валюта: Доллар

switch (newCurrency) {

case 1:

conversionRate = usdRate / eurRate;

break;

case 2:

conversionRate = 1.0;

break;

case 3:

conversionRate = usdRate / cnyRate;

break;

case 4:

conversionRate = usdRate / gbpRate;

break;

case 5:

conversionRate = usdRate / rubRate;

break;

default:

cout << "Неверный выбор финальной валюты" << endl;

break;

}

break;

case 3: // Исходная валюта: Юань

switch (newCurrency) {

case 1:

conversionRate = cnyRate / eurRate;

break;

case 2:

conversionRate = cnyRate / usdRate;

break;

case 3:

conversionRate = 1;

break;

case 4:

conversionRate = cnyRate / gbpRate;

break;

case 5:

conversionRate = cnyRate / rubRate;

break;

default:

cout << "Неверный выбор финальной валюты" << endl;

break;

}

break;

case 4: // Исходная валюта: Фунт

switch (newCurrency) {

case 1:

conversionRate = gbpRate / eurRate;

break;

case 2:

conversionRate = gbpRate / usdRate;

break;

case 3:

conversionRate = gbpRate / cnyRate;

break;

case 4:

conversionRate = 1;

break;

case 5:

conversionRate = gbpRate / rubRate;

break;

default:

cout << "Неверный выбор финальной валюты" << endl;

break;

}

break;

case 5: // Исходная валюта: Рубль

switch (newCurrency) {

case 1:

conversionRate = rubRate / eurRate;

break;

case 2:

conversionRate = rubRate / usdRate;

break;

case 3:

conversionRate = rubRate / cnyRate;

break;

case 4:

conversionRate = rubRate / gbpRate;

break;

case 5:

conversionRate = 1;

break;

default:

cout << "Неверный выбор финальной валюты" << endl;

break;

}

break;

default:

cout << "Неверный выбор изначальной валюты" << endl;

break;

};

// Вычисление конвертированной суммы

double convertedAmount = amount \* conversionRate;

cout << "Результат перевода: " << convertedAmount << endl << endl;

}

};

#endif /\* Bank\_h \*/

SavingAccount.h

//

// SavingAccount\_h

// CPP\_Lab6

//

// Created by Danil Morozov on 16.05.2023.

//

#ifndef SavingAccount\_h

#define SavingAccount\_h

#include "Bank.h"

#include "helper.h"

using namespace std;

class SavingAccount: public BankAccount {

private:

double interestRate;

bool isLocked;

double savedBalance;

public:

SavingAccount(double interestRate, bool isLocked, double savedBalance, string accountNumber, double balance) : BankAccount(accountNumber, balance) {

this->interestRate = interestRate;

this->isLocked = isLocked;

this->savedBalance = savedBalance;

}

// Собственные методы

double getInterestRate() const {

return this->interestRate;

}

double getSavedBalance() {

return this->savedBalance;

}

void addToSavedBalance(double amount) {

this-> savedBalance += amount;

}

void decFromSavedBalance(double amount) {

this-> savedBalance -= amount;

}

void setInterestRate(double rate) {

interestRate = rate;

}

bool isAccountLocked() {

return this->isLocked;

}

void lockAccount() {

isLocked = !isLocked;

}

void calculateInterest() {

double interest = savedBalance \* (interestRate / 100);

cout << "Процентное начисление: " << interest << endl;

savedBalance += interest;

}

// Переопределенные методы из родительского класса

void displayAccountInfo() override {

cout << "Информация о сберегательном счете" << endl;

cout << "Номер счета: " << getAccName() << endl;

cout << "Баланс: " << getBalance() << " RUB" << endl;

cout << "Баланс счета вклада: " << getSavedBalance() << " RUB" << endl;

cout << "Процент начисления: " << getInterestRate() << "%" << endl;

cout << "Статус счета: " << (isAccountLocked() ? "Заблокирован" : "Разблокирован") << endl;

}

void deposit(double amount) override {

if (isLocked) {

cout << "Счет заблокирован. Операция депозита невозможна." << endl;

} else {

if (getBalance() < amount) {

cout << "Вывод средств невозможен, на счету недостаточно денег!" << endl;

} else {

decFromBalance(amount);

addToSavedBalance(amount);

}

}

}

void withdraw(double amount) override {

if (isLocked) {

cout << "Счет заблокирован. Операция депозита невозможна." << endl;

} else {

if (savedBalance < amount) {

cout << "Вывод средств невозможен, на сберегательном счету недостаточно денег!" << endl;

} else {

decFromSavedBalance(amount);

addToBalance(amount);

}

}

}

void update() override {

string newValue;

int state;

cout << "1: Изменить номер счета" << endl;

cout << "2: Изменить баланс" << endl;

cout << "3: Заблокировать/Разблокировать сберегательный счет" << endl;

cout << "4: Указать процентную ставку" << endl;

cout << "5: Провести процентное начисление" << endl;

cout << "6: Депозит средств на сберегательный счет" << endl;

cout << "7: Вывести средства со сберегательного счета" << endl;

state = input();

switch (state)

{

case 1:

cout << "Введите значение: ";

cin >> newValue;

setAccName(newValue);

break;

case 2:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

setBalance(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 3:

lockAccount();

break;

case 4:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

setInterestRate(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 5:

calculateInterest();

break;

case 6:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

deposit(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 7:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

withdraw(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

default:

cout<< "Ошибка при выполнении операции!" << endl << endl;

break;

}

}

};

#endif /\* SavingAccount\_h \*/

CreditAccount.h

//

// CreditAccount.h

// CPP\_Lab6

//

// Created by Danil Morozov on 17.05.2023.

//

#ifndef CreditAccount\_h

#define CreditAccount\_h

#include "Bank.h"

#include "helper.h"

using namespace std;

class CreditAccount : public BankAccount {

private:

double creditAmount;

bool hasCredit;

public:

CreditAccount(double creditAmount, bool hasCredit, string accountNumber, double balance) : BankAccount(accountNumber, balance) {

this -> creditAmount = creditAmount;

this -> hasCredit = hasCredit;

}

bool getHasCredit() {

if (creditAmount == 0) {

this->hasCredit = false;

}

return this->hasCredit;

}

double getCreditAmount() {

return this-> creditAmount;

}

void takeCredit(double amount) {

this -> creditAmount = (amount \* 1.10);

}

void setCreditAmount(double amount) {

this -> creditAmount = amount;

}

void pay(double amount) {

this -> creditAmount -= amount;

}

void setCredit() {

this -> hasCredit = true;

}

// Перекрытые методы

void displayAccountInfo() override {

cout << "Информация о кредитном счете" << endl;

cout << "Номер счета: " << getAccName() << endl;

cout << "Баланс: " << getBalance() << " RUB" << endl;

cout << "Ваш кредитный процент - 10%" << endl;

cout << "Статус кредита: " << (getHasCredit() ? "Имеется" : "Не имеется") << endl;

cout << "Размер заема: " << getCreditAmount() << " RUB" << endl;

}

void deposit(double amount) override {

if (hasCredit) {

if (amount > getBalance()) {

cout << "Недостаточно средств для пополнения кредитного счета" << endl;

} else {

if (amount == creditAmount) {

setCreditAmount(0);

decFromBalance(amount);

hasCredit = false;

} else if (amount < creditAmount) {

decFromBalance(amount);

pay(amount);

} else {

cout << "Вы не можете внести платеж превышающий заем" << endl;

};

};

} else {

cout << "У вас нет задолжностей" << endl;

}

}

void withdraw(double amount) override {

if (getHasCredit()){

cout << "Вы не можете взять заем, имеется задолженность!" << endl;

} else {

if (amount >= 50000) {

cout << "Вы не можете взять заем выше 50000 RUB" << endl;

} else {

setCredit();

takeCredit(amount);

addToBalance(amount);

cout << "Кредит на сумму " << amount << " RUB был взят" << endl;

};

};

}

void update() override {

string newValue;

int state;

cout << "1: Изменить номер счета" << endl;

cout << "2: Изменить баланс" << endl;

cout << "3: Проверить наличие кредита" << endl;

cout << "4: Взять кредит" << endl;

cout << "5: Депозит средств на кредитный счет" << endl;

state = input();

switch (state)

{

case 1:

cout << "Введите значение: ";

cin >> newValue;

setAccName(newValue);

break;

case 2:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

setBalance(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 3:

cout << "Статус кредита: " << (getHasCredit() ? "Имеется" : "Не имеется") << endl;

break;

case 4:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

withdraw(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

case 5:

cout << "Введите значение: ";

cin >> newValue;

if (check(newValue)) {

deposit(stod(newValue));

}

else {

cout<< "Ошибка при выполнении операции!" << endl << endl;

}

break;

default:

cout<< "Ошибка при выполнении операции!" << endl << endl;

break;

}

}

};

#endif /\* CreditAccount\_h \*/

main.cpp

//

// main.cpp

// CPP\_Lab6

//

// Created by Danil Morozov on 12.05.2023.

//

#include <iostream>

#include "Bank.h"

#include "SavingAccount.h"

#include "CreditAccount.h"

#include "helper.h"

using namespace std;

int main() {

bool flag = false;

string accName;

bool YN;

double bal, savBal, creBal, procVal;

string b, m, o, e, g;

int y, p, type, msg;

//заренее создаются указатели на классы, которые, в зависимости от выбора, будут инициализированы

BankAccount\* bank = nullptr;

header\_start();

//пока не будет выбран класс

while (!flag) {

type = input();

switch (type)

{

case 1:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

bank = new BankAccount(accName, bal);

flag = true;

break;

case 2:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

cout << "Введите процентную ставку" << endl;

procVal = inputDouble();

cout << "Аккаунт будет с блокировкой? (Y/N)" << endl;

YN = inputYesOrNo();

cout << "Введите исходный баланс на сберегательном счету" << endl;

savBal = inputDouble();

bank = new SavingAccount(procVal, YN, savBal, accName, bal);

flag = true;

break;

case 3:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

cout << "Аккаунт будет иметь кредит? (Y,N)" << endl;

YN = inputYesOrNo();

if (YN) {

cout << "Размер кредита" << endl;

creBal = inputDouble();

} else {

creBal = 0;

};

bank = new CreditAccount(creBal\*1.10, YN, accName, bal);

flag = true;

break;

default:

cout << "Произошла ошибка!" << endl;

flag = false;

break;

}

}

flag = false;

header();

//цикл обработки сообщений для работы с методами класса

while (!flag) {

cout << "Выберите метод: ";

msg = input();

cout << endl;

//не смотря на то, что объекты имеют разные классы - методы их одинаковы

//это связано с тем, что некоторые методы являются переопределенными, а некоторые

// унаследованными от родительского класса

switch (msg)

{

case 1: //вывод всех полей

bank->displayAccountInfo();

break;

case 2:

bank->update(); //обновление одного поля

break;

case 3:

bank->calcProcentage(); //расситать проценты

break;

case 4:

bank->currencyCheck(); //перевод валют

break;

case 5:

//переопределение типов

//нужно это для того, чтобы можно было обращатся к собственным методам каждого класса класса

if (type == 1) {

//собственные методы класса BankAccount

cout << "Номер счета: " << bank->getAccName() << endl;

cout << "Баланс: " << bank->getBalance() << " RUB" << endl;

}

if (type == 2) {

bool locked = ((SavingAccount\*)bank)->isAccountLocked();

cout << "Процентная ставка: " << ((SavingAccount\*)bank)->getInterestRate() << endl;

cout << "Статус счета: " << (locked ? "Заблокирован" : "Разблокирован") << endl;

cout << "Баланс сберегательного счета: " << ((SavingAccount\*)bank)->getSavedBalance() << endl;

}

if (type == 3) {

//собственные методы класса CreditAccount

cout << "Наличие кредита: " << (((CreditAccount\*)bank)->getHasCredit() ? "Имеется" : "Не имеется") << endl;

cout << "Размер кредита: " << ((CreditAccount\*)bank)->getCreditAmount() << endl;

}

break;

case 6:

flag = false;

header\_start();

type = input();

switch (type)

{

case 1:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

bank = new BankAccount(accName, bal);

flag = false;

header();

break;

case 2:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

cout << "Введите процентную ставку" << endl;

procVal = inputDouble();

cout << "Аккаунт будет с блокировкой? (Y/N)" << endl;

YN = inputYesOrNo();

cout << "Введите исходный баланс на сберегательном счету" << endl;

savBal = inputDouble();

bank = new SavingAccount(procVal, YN, savBal, accName, bal);

flag = false;

header();

break;

case 3:

cout << "Введите номер счета" << endl;

cin >> accName;

cout << "Введите исходный баланс" << endl;

bal = inputDouble();

cout << "Аккаунт будет иметь кредит? (Y,N)" << endl;

YN = inputYesOrNo();

if (YN) {

cout << "Размер кредита" << endl;

creBal = inputDouble();

} else {

creBal = 0;

};

bank = new CreditAccount(creBal\*1.10, YN, accName, bal);

flag = false;

header();

break;

default:

cout << "Произошла ошибка!" << endl;

flag = false;

break;

}

break;

case 7:

//очищение памяти происходит по такой логике, потому что не нужно очищать те указатели, которые не выбраны

//то есть, очищается указатель только того класса, с которым велась работа

delete bank;

flag = true;

break;

default:

cout << "Произошла ошибка!" << endl;

break;

}

}

};

helper.h

//

// helper.h

// CPP\_Lab6

//

// Created by Danil Morozov on 16.05.2023.

//

#ifndef helper\_h

#define helper\_h

#include <iostream>

#include <string>

using namespace std;

bool checkDouble(string inp) {

for (char c : inp) {

if (!(c == '.' or isdigit(c) or c == '-')) {

return false;

}

}

return true;

}

bool check(string inp) {

for (char c : inp) {

if (!isdigit(c) or c == '-') {

return false;

}

}

return true;

}

int checkYN(char inp) {

if (!(inp == 'Y' or inp == 'N' or inp == 'y' or inp == 'n')) {

return 0;

} else {

if (inp == 'Y' or inp == 'y') {

return 1;

} else {

return 2;

};

};

}

int input() {

string inp;

cin >> inp;

while (!check(inp)) {

cout << "Ошибка ввода! \n" << "Повторите: ";

cin.clear();

cin >> inp;

}

return stoi(inp);

}

double inputDouble() {

string inp;

cin >> inp;

while (!checkDouble(inp)) {

cout << "Ошибка ввода! \n" << "Повторите: ";

cin.clear();

cin >> inp;

}

return stod(inp);

}

bool inputYesOrNo() {

char inp;

cin >> inp;

while (checkYN(inp) == 0) {

cout << "Ошибка ввода! \n" << "Повторите: ";

cin.clear();

cin >> inp;

}

if (inp == 'Y' or inp == 'y') {

return true;

} else {

return false;

};

}

void header\_start() {

cout << "Выберите класс, с которым хотите работать:" << endl;

cout << "1) Банк" << endl;

cout << "2) Сберегательный счет" << endl;

cout << "3) Кредитный аккаунт" << endl;

}

void header() {

cout << endl << "1) Вывести поля объекта" << endl;

cout << "2) Изменить поле объекта" << endl;

cout << "3) Расчитать проценты" << endl;

cout << "4) Перевести валюты" << endl;

cout << "5) Вывести собственные поля класса" << endl;

cout << "6) Выйти в меню" << endl;

cout << "7) Закрыть программу" << endl;

}

#endif /\* helper\_h \*/

**Экранные формы:**

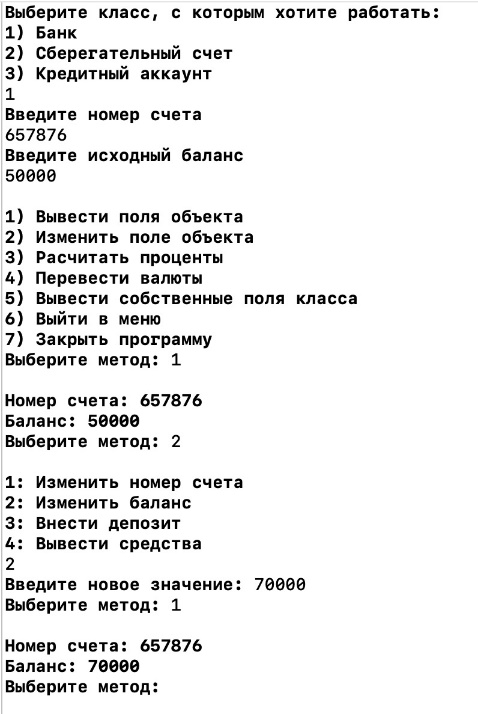


Рис. 2 – выбор родительского класса, поля, изменение полей

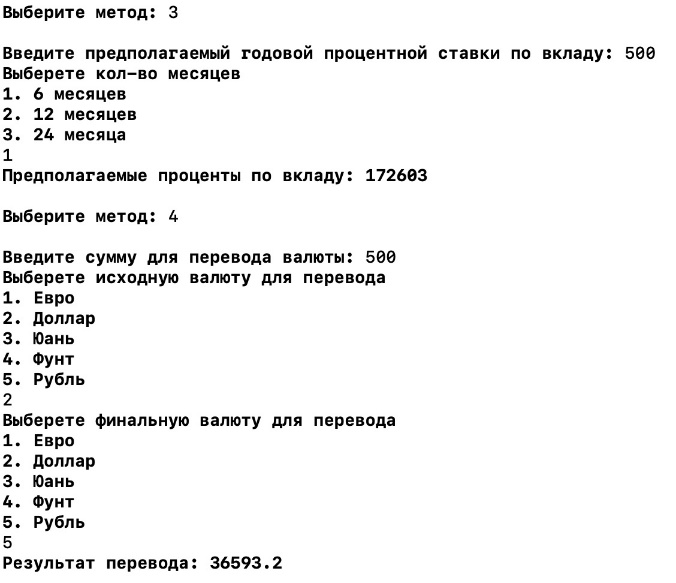


Рис. 3 – глобальные методы родительского класса

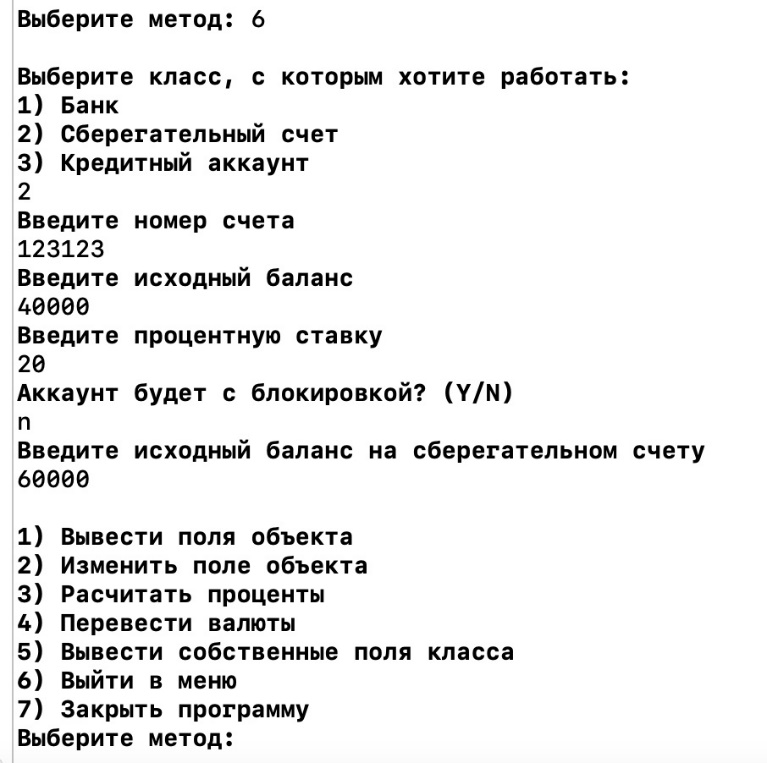


Рис. 4 – переход к дочернему классу SavingAccount

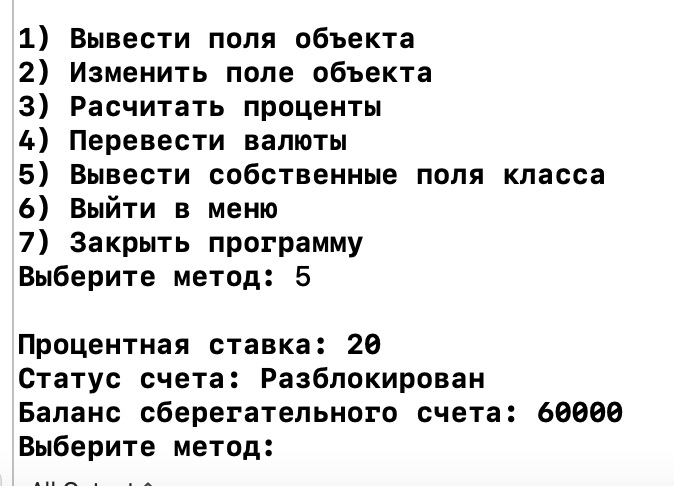


Рис. 5 – собственные поля класса SavingAccount

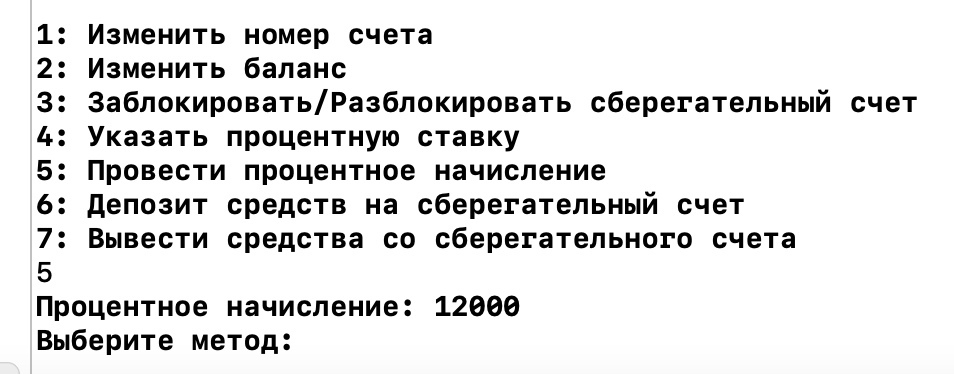


Рис. 6 – пример работы собственного метода calculateInterest()

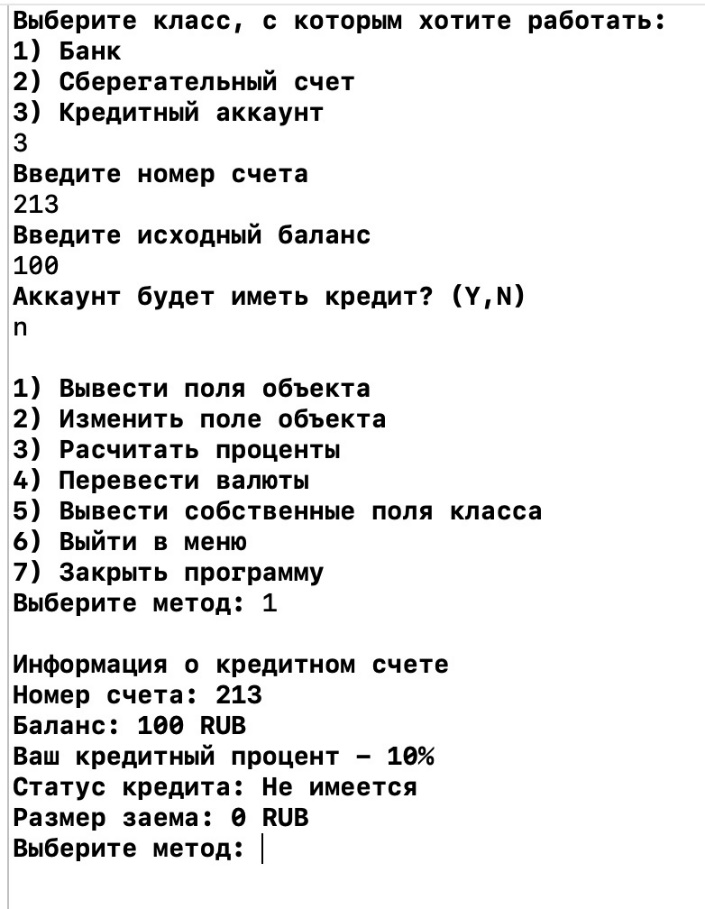


Рис. 7 – инициализация объекта класса CreditAccount и вывод его полей

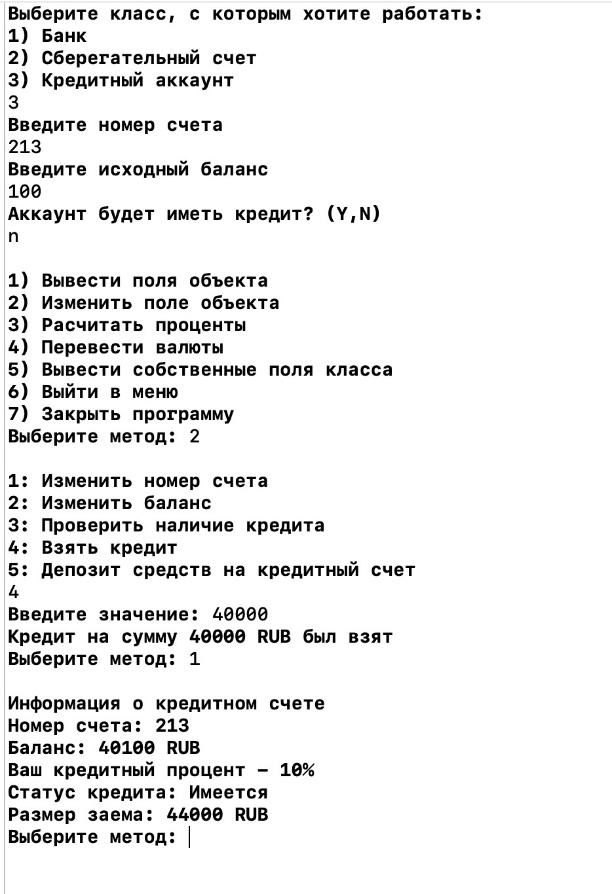


Рис. 8 – изменение полей объекта класса CreditAccount, вызов метода takeCredit

**Вывод:**

В результате выполнения данной лабораторной работы было разработано консольное приложение, которое демонстрирует три ключевых принципа объектно-ориентированного программирования (ООП): инкапсуляцию, наследование и полиморфизм.

Принцип инкапсуляции в ООП подразумевает скрытие методов и полей объекта внутри самого объекта. Это позволяет обеспечить доступ к данным только через определенные интерфейсы и методы, что способствует контролю и безопасности данных.

Принцип наследования позволяет создавать новые классы, которые наследуют все доступные (public) поля и методы родительского класса. При этом класс-наследник может добавлять свои собственные поля и методы. Наследование способствует повторному использованию кода, а также упрощает его расширение и изменение.

Принцип полиморфизма предполагает возможность переопределения унаследованных методов родительского класса в классе-наследнике. Это позволяет использовать один и тот же метод для объектов разных классов, при этом они будут вести себя соответствующим образом в зависимости от своего типа. Полиморфизм способствует гибкости и расширяемости программы.

Также следует отметить, что в ООП применяется неявная типизация, которая позволяет передавать объекты дочернего класса в качестве аргументов, где ожидается объект родительского класса. Это возможно благодаря наследованию, где дочерний класс автоматически является экземпляром родительского класса.

Неявная типизация родительского класса к дочернему предоставляет гибкость в использовании классов и методов в иерархии наследования. Когда объект дочернего класса передается в метод, который ожидает объект родительского класса, он все равно будет вести себя соответствующим образом, используя переопределенные методы и поля в дочернем классе. Это позволяет создавать более обобщенные методы, которые могут работать с объектами разных классов, но имеют общий интерфейс.

Таким образом, принципы наследования и полиморфизма, в сочетании с неявной типизацией, позволяют создавать гибкие и расширяемые программы, где классы могут использоваться взаимозаменяемо, а методы могут быть переопределены и адаптированы под конкретные потребности дочерних классов.

В данной лабораторной работе было продемонстрировано использование этих принципов через создание и инициализацию классов, а также вызов их методов. Это позволило наглядно увидеть принципы наследования и полиморфизма в действии.