МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА компьютерных технологий и программной инженерии |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | Шумова Е.О. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «Наследование классов, базовый класс, производный класс» |
| по дисциплине: Объектно-ориентированное программирование |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 |  |  |  | Андреев Д.И. |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2019/3781 | |  |  |  |

Санкт-Петербург 2021

Оглавление

[1. Условие 3](#_Toc70910120)

[2. Листинг программы 3](#_Toc70910121)

[3. Результаты работы программы. 7](#_Toc70910122)

[4. Вывод 7](#_Toc70910123)

1. Условие

Вариант 1. Создать класс Money представляющий количество банкнот достоинством 10, 50, 100, 500, 1000, 5000. Предусмотреть метод summa, для вычисления общей суммы. На основе класса Money создать класс Bankomat предусматривающий снятие любой возможной суммы, пополнение запаса. В программе предусмотреть возможность неограниченного количества пополнения и снятия со счета.

1. Листинг программы

Файл Money.h

#pragma once

#include <map>

class Money

{

public:

enum banknoteDenomination

{

Undefined = 0,

Denomination\_10 = 10,

Denomination\_50 = 50,

Denomination\_100 = 100,

Denomination\_500 = 500,

Denomination\_1000 = 1000,

Denomination\_5000 = 5000,

};

Money();

Money(int d10, int d50, int d100, int d500, int d1000, int d5000);

int summa();

protected:

int count(banknoteDenomination type);

void increeceCount(banknoteDenomination type, int delta);

int decreeceCount(banknoteDenomination type, int delta);

private:

std::map<banknoteDenomination, int> m\_banknoteCount;

};

Файл Money.cpp

#include "Money.h"

#include <cmath>

Money::Money()

: m\_banknoteCount()

{

m\_banknoteCount[Denomination\_10] = 0;

m\_banknoteCount[Denomination\_50] = 0;

m\_banknoteCount[Denomination\_100] = 0;

m\_banknoteCount[Denomination\_500] = 0;

m\_banknoteCount[Denomination\_1000] = 0;

m\_banknoteCount[Denomination\_5000] = 0;

}

Money::Money(int d10, int d50, int d100, int d500, int d1000, int d5000)

: m\_banknoteCount()

{

m\_banknoteCount[Denomination\_10] = d10;

m\_banknoteCount[Denomination\_50] = d50;

m\_banknoteCount[Denomination\_100] = d100;

m\_banknoteCount[Denomination\_500] = d500;

m\_banknoteCount[Denomination\_1000] = d1000;

m\_banknoteCount[Denomination\_5000] = d5000;

}

int Money::summa()

{

int summ = 0;

for (auto banknotesPair : m\_banknoteCount)

summ += banknotesPair.first \* banknotesPair.second;

return summ;

}

int Money::count(banknoteDenomination type)

{

return m\_banknoteCount[type];

}

void Money::increeceCount(banknoteDenomination type, int delta)

{

m\_banknoteCount[type] += delta;

}

int Money::decreeceCount(banknoteDenomination type, int delta)

{

m\_banknoteCount[type] -= delta;

if (m\_banknoteCount[type] < 0) {

int remains = std::abs(m\_banknoteCount[type]);

m\_banknoteCount[type] = 0;

return remains;

}

return 0;

}

Файл Bankomat.h

#pragma once

#include "Money.h"

class Bankomat : public Money

{

public:

Bankomat();

Bankomat(int d10, int d50, int d100, int d500, int d1000, int d5000);

void deposit(int count, Money::banknoteDenomination banknoteType);

void withdraw(int count);

private:

bool checkBanknoteDenomination(Money::banknoteDenomination type);

};

Файл Bankomat.cpp

#include "Bankomat.h"

#include <iostream>

Bankomat::Bankomat()

: Money()

{

}

Bankomat::Bankomat(int d10, int d50, int d100, int d500, int d1000, int d5000)

: Money(d10, d50, d100, d500, d1000, d5000)

{

}

void Bankomat::deposit(int count, Money::banknoteDenomination banknoteType)

{

if (count < 0) {

std::cerr << "cannot depoosit negative money count\n";

return;

}

else if (count % 10 != 0) {

std::cerr << "it is not possible to withdraw an amount that is not a multiple of ten\n";

return;

}

else if (count < 0) {

std::cerr << "it is not possible to deposit negative amount\n";

return;

}

else if (!checkBanknoteDenomination(banknoteType)) {

std::cerr << "cannot recognise banknote denomination\n";

return;

}

increeceCount(banknoteType, count);

}

void Bankomat::withdraw(int amount)

{

if (summa() == 0) {

std::cerr << "Bankomat is empty\n";

return;

}

else if (amount % 10 != 0) {

std::cerr << "it is not possible to withdraw an amount that is not a multiple of ten\n";

return;

}

else if (amount < 0) {

std::cerr << "it is not possible to withdraw negative amount\n";

return;

}

int remains = amount;

int d10 = 0;

int d50 = 0;

int d100 = 0;

int d500 = 0;

int d1000 = 0;

int d5000 = 0;

while (remains != 0) {

if (Denomination\_5000 <= remains && count(Denomination\_5000) > 0) {

int banknotesCount = remains / Denomination\_5000;

int remainsCount = decreeceCount(Denomination\_5000, banknotesCount);

d5000 = (banknotesCount - remainsCount) \* Denomination\_5000;

remains -= (banknotesCount - remainsCount) \* Denomination\_5000;

}

else if (Denomination\_1000 <= remains && count(Denomination\_1000) > 0) {

int banknotesCount = remains / Denomination\_1000;

int remainsCount = decreeceCount(Denomination\_1000, banknotesCount);

d1000 = (banknotesCount - remainsCount) \* Denomination\_1000;

remains -= (banknotesCount - remainsCount) \* Denomination\_1000;

}

else if (Denomination\_500 <= remains && count(Denomination\_500) > 0) {

int banknotesCount = remains / Denomination\_500;

int remainsCount = decreeceCount(Denomination\_500, banknotesCount);

d500 = (banknotesCount - remainsCount) \* Denomination\_500;

remains -= (banknotesCount - remainsCount) \* Denomination\_500;

}

else if (Denomination\_100 <= remains && count(Denomination\_100) > 0) {

int banknotesCount = remains / Denomination\_100;

int remainsCount = decreeceCount(Denomination\_100, banknotesCount);

d100 = (banknotesCount - remainsCount) \* Denomination\_100;

remains -= (banknotesCount - remainsCount) \* Denomination\_100;

}

else if (Denomination\_50 <= remains && count(Denomination\_50) > 0) {

int banknotesCount = remains / Denomination\_50;

int remainsCount = decreeceCount(Denomination\_50, banknotesCount);

d50 = (banknotesCount - remainsCount) \* Denomination\_50;

remains -= (banknotesCount - remainsCount) \* Denomination\_50;

}

else if (Denomination\_10 <= remains && count(Denomination\_10) > 0) {

int banknotesCount = remains / Denomination\_10;

int remainsCount = decreeceCount(Denomination\_10, banknotesCount);

d10 = (banknotesCount - remainsCount) \* Denomination\_10;

remains -= (banknotesCount - remainsCount) \* Denomination\_10;

}

else break;

}

std::cout << "withdraw:\n"

<< d10 << " banknotes denomination of 10\n"

<< d50 << " banknotes denomination of 50\n"

<< d100 << " banknotes denomination of 100\n"

<< d500 << " banknotes denomination of 500\n"

<< d1000 << " banknotes denomination of 1000\n"

<< d5000 << " banknotes denomination of 5000\n"

<< remains << " did not withdraw\n\n";

}

bool Bankomat::checkBanknoteDenomination(Money::banknoteDenomination type)

{

return type == Denomination\_10

|| type == Denomination\_50

|| type == Denomination\_100

|| type == Denomination\_500

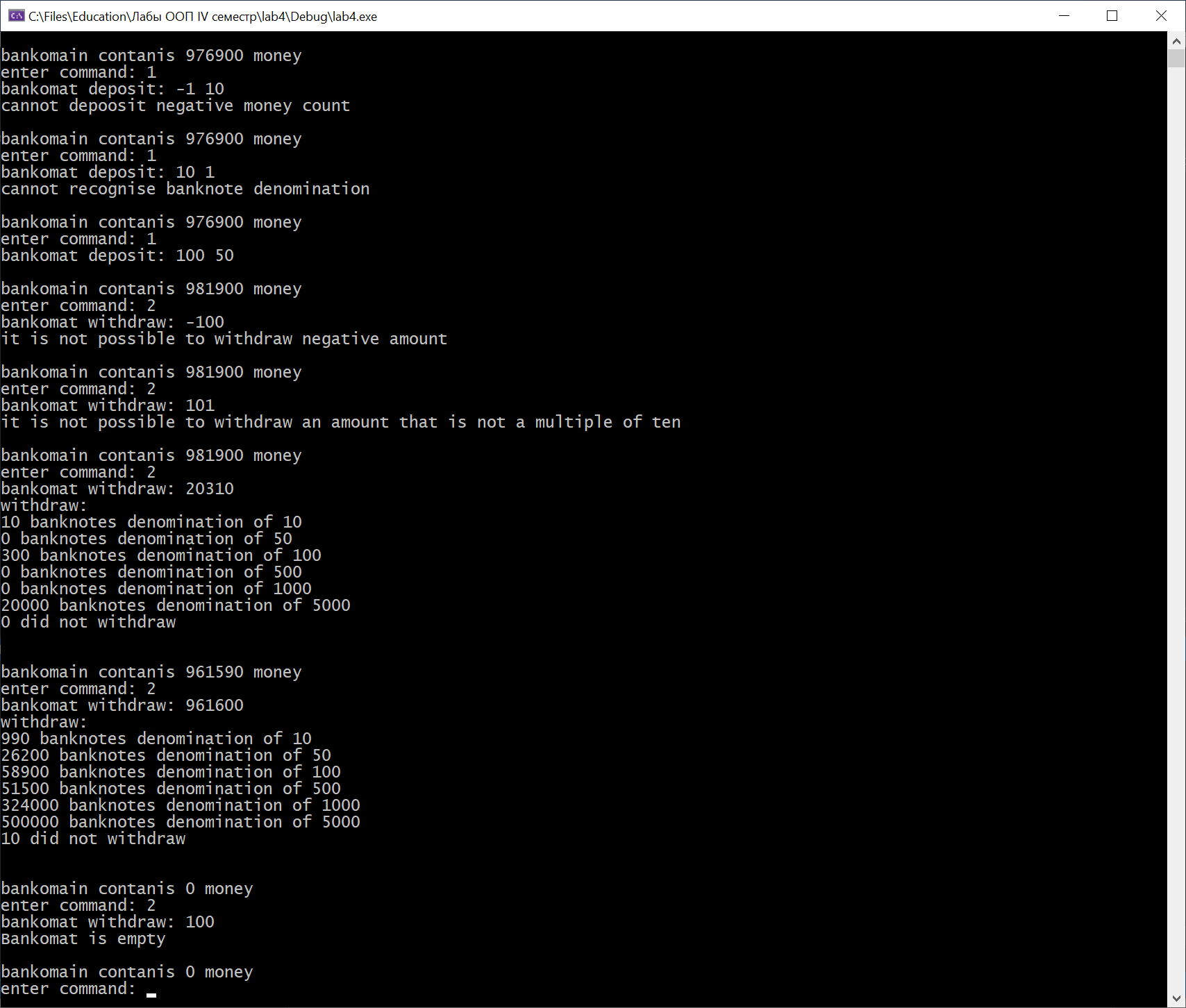
|| type == Denomination\_1000

|| type == Denomination\_5000;

}

1. Результаты работы программы.

Результат работы тестовой программы:



Тестовая программа работает следующим образом. На вход принимается команда: 1 – положить купюры (принимает на вход первым числом количество купюр, вторым – их достоинство), 2 – выдать купюры (результат работы количество выданных купюр по номиналу и сумма, которая не была выдана.

1. Вывод

В ходе лабораторной работы были разработаны классы, описывающие работу банкомата. Использование наследования позволило унаследовать интерфейс базового класса и расширить его функционал. Недостатки данной реализации в том, что в данном случае реализовано отношение является, но банкомат не является деньгами. Данный недостаток решается заменой отношения является отношением содержит, что позволит расширить возможности использования разработанных классов.