БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Международный Институт Дистанционного Образования

Кафедра “Информационные системы и технологии”

Лабораторная работа по дисциплине «ООП»

Тема работы:

**«ООП на С++»**

Выполнил:   
студент 2 курса, гр. 41703120  
Реут Владислав Леонидович  
№ зачетной книжки 417031208

Проверил: Радкевич А.С.

Минск 2022**ВВЕДЕНИЕ**

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

С++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на С++

С++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на С++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на С++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на С++ в приложения под эти платформы.

В 1979-80 годах Бьерн Страуструп разработал расширение к языку Си - "Си с классами". В 1983 язык был переименован в С++.

В 1985 году была выпущена первая коммерческая версия языка С++, а также первое издание книги "Языка программирования C++", которая представляла первое описание этого языка при отсутствии официального стандарта.

В 1989 была выпущена новая версия языка C++ 2.0, которая включала ряд новых возможностей. После этого язык развивался относительно медленно вплоть до 2011 года. Но при этом в 1998 году была предпринята первая попытка по стандартизации языка организацией ISO (International Organiztion for Standartization). Первый стандарт получил название ISO/IEC 14882:1998 или сокращенно С++98. В дальнейшем в 2003 была издана новая версия стандарта C++03.

**Лабораторная работа №1**

**КЛАСС, СОЗДАНИЕОБЪЕКТ АКЛАССА. ПОНЯТИЕ ИНКАПСУЛЯЦИИ**

Цель работы: получить навыки проектирования простейших классов. Научиться создавать объекты класса. Освоить принцип инкапсуляции.

**Задание к работе**

1.Выбрать предметную область согласно варианту индивидуального задания.

2.Спроектировать класс для выбранной предметной области.

3.Предусмотреть наличие у объекта полей, методов и свойств.

4.Предусмотреть наличие свойств только для записи.

5.Предметная область: Банк. В классе хранить информацию о наименовании банка, числе вкладов, размере вклада (все вклады одинаковые), размере процентной ставки. Реализовать метод для подсчета общей выплаты по процентам.

Код:

Содержимое заголовочного файла:

class Bank

{

public:

Bank();

Bank(string n, int nDep, double dAmount, double iRate);

string getName();

int getNumDep();

double getDepAmount();

double getInterRate();

double getInterPay();

~Bank();

private:

string name;// название учреждения

int numDep;// количество депозитов

double depAmount, interRate;// сумма депозита, процентная ставка

};

Содержимое основного файла:

Bank::Bank(string n, int nDep, double dAmount, double iRate)

: name(n), numDep(nDep), depAmount(dAmount), interRate(iRate) {}

Bank::Bank()

{

name = "";

numDep = 0;

depAmount = 0.0;

interRate = 0.0;

}

string Bank::getName()

{

return name;

}

int Bank::getNumDep()

{

return numDep;

}

double Bank::getDepAmount()

{

return depAmount;

}

double Bank::getInterRate()

{

return interRate;

}

double Bank::getInterPay()

{

return ((depAmount \* interRate \* 273 / 365) / 100);

}

Bank::~Bank() {}

Содержимое main файла:

Bank b("Priorbank", 130, 350000, 4.7);

cout << b.getInterPay() << endl;

Результат работы:



**Лабораторная работа №2**

**КОНСТРУКТОРЫ. СТАТИЧЕСКИЕ ЧЛЕНЫК ЛАССА.**

**ШАБЛОН ПРОЕКТИРОВАНИЯ SINGLETON**

Цель работы: изучить работу и назначение конструкторов.

Освоить возможности членов класса с модификатором static. Ознакомиться с шаблоном проектирования Singleton.

**Задание к работе:**

1.Спроектировать классы для выбранной предметной области.

2.Применить к одному из классов шаблон проектирования Singleton.

**Индивидуальные задания:**

5.Аэропорт–Взлетная полоса.

Код:

Содержимое заголовочного файла:

class Runway

{

public:

Runway();

Runway(int rwNum);

Runway(double l, double w, string p, int rwNum);

double getLength();

double getWidth();

string getPavement();

void setRunwayNumber(int rwNum);

int getRunwayNumber();

~Runway();

private:

double length, width;// длинна, ширина

int runwayNumber;

string pavement;// дорожное покрытие

};

class Airport

{

string airportName;

Runway runway;// переменная взлетной полосы

static Airport\* instAirport; // статический указатель типа Эйрпот

int boardNumber;// номер борта

string destination;// пункт назначения

int arrivalHour, arrivalMinut, departureHour, departureMinut;// время прибытия, время убытия.

public:

static Airport\* InstanceAirport();// статический метод для выделения кдинственного экземпляра(Singleton)

void setAirportName(string \_airportName);

void setBoardNumber(int bNum);

void setDestination(string d\_tion);

void setArrivalHour(int arHour);

void setDepartureHour(int deHour);

void setArrivalMinut(int arMinut);

void setDepartureMinut(int deMinut);

void setRunway(int \_runway);

string getAirportName();

int getBoardNumber();

string getDestination();

int getArrivalHour();

int getDepartureHour();

int getArrivalMinut();

int getDepartureMinut();

Runway getRunway();

void print();

protected:

Airport();

~Airport();

};

Содержимое основного файла:

// Runway

Runway::Runway()

{

length = 300.0;

width = 10.0;

pavement = "";

runwayNumber = 11;

}

Runway::Runway(double l, double w, string p, int rwNum)

: length(l), width(w), pavement(p), runwayNumber(rwNum){}

Runway::Runway(int rwNum)

{

length = 300.0;

width = 10.0;

pavement = "";

runwayNumber = rwNum;

}

double Runway::getLength()

{

return length;

}

double Runway::getWidth()

{

return width;

}

string Runway::getPavement()

{

return pavement;

}

void Runway::setRunwayNumber(int rwNum)

{

runwayNumber = rwNum;

}

int Runway::getRunwayNumber()

{

return runwayNumber;

}

Runway::~Runway() {}

// Airport

Airport\* Airport::instAirport = nullptr;

Airport::Airport()

{

airportName = "Minsk2";

boardNumber = 77421;

destination = "Stambul";

departureHour = 17;

departureMinut = 12;

arrivalHour = 20;

arrivalMinut = 51;

//runway = Runway(17);

runway.setRunwayNumber(13);

}

Airport\* Airport::InstanceAirport()

{

if (instAirport == nullptr)

{

instAirport = new Airport();

return instAirport;

}

else

{

return nullptr;

}

}

void Airport::setAirportName(string \_airportName)

{

airportName = \_airportName;

}

void Airport::setBoardNumber(int bNum)

{

boardNumber = bNum;

}

void Airport::setDestination(string d\_tion)

{

destination = d\_tion;

}

void Airport::setArrivalHour(int arHour)

{

arrivalHour = arHour;

}

void Airport::setDepartureHour(int deHour)

{

departureHour = deHour;

}

void Airport::setArrivalMinut(int arMinut)

{

arrivalMinut = arMinut;

}

void Airport::setDepartureMinut(int deMinut)

{

departureMinut = deMinut;

}

void Airport::setRunway(int \_runway)

{

runway.setRunwayNumber(\_runway);

}

string Airport::getAirportName()

{

return airportName;

}

int Airport::getBoardNumber()

{

return boardNumber;

}

string Airport::getDestination()

{

return destination;

}

int Airport::getArrivalHour()

{

return arrivalHour;

}

int Airport::getDepartureHour()

{

return departureHour;

}

int Airport::getArrivalMinut()

{

return arrivalMinut;

}

int Airport::getDepartureMinut()

{

return departureMinut;

}

Runway Airport::getRunway()

{

return runway.getRunwayNumber();

}

void Airport::print()

{

cout << getAirportName();

}

Airport::~Airport()

{

delete instAirport;

}

Содержимое main файла:

// Создание обЪекта Singleton

Airport\* Minsk1 = Airport::InstanceAirport();

if (Minsk1 != nullptr)

{

cout << "Airport name: " << Minsk1->getAirportName() << endl

<< "Destination: " << Minsk1->getDestination() << endl

<< "Departure time: " << Minsk1->getDepartureHour() << ":" << Minsk1->getDepartureMinut() << endl

<< "Arrival time: " << Minsk1->getArrivalHour() << ":" << Minsk1->getArrivalMinut() << endl

<< "Runway number: " << Minsk1->getRunway().getRunwayNumber() << endl;

}

// Попытка создание 2го объекта Singleton

Airport\* Minsk2 = Airport::InstanceAirport();

if (Minsk2 != nullptr)

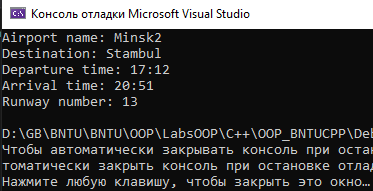
{

Minsk2->setDestination("Minsk2");

cout << Minsk2->getDestination();

}

Результат работы:



Как видно 2й объект Singleton не создался.

**Лабораторная работа №3**

**ИСПОЛЬЗОВАНИЕ КОЛЛЕКЦИЙ**

Цель работы: получить навыки проектирования приложения, состоящего из нескольких взаимосвязанных классов.

**Задание к работе**

1.Для заданной предметной области спроектировать программную структуру, состоящуюиз3–5классов.

2.Предусмотреть использование типа данных – перечисление.

3.Ввод/вывод должен быть реализован вне проектируемого класса.

**Индивидуальные задания**

5.Предметная область: Банк. Информационная система банка хранит описание процентов по различным вкладам. Система хранит информацию о вкладчиках и сделанных ими вкладах. Каждый клиент может поместить в банк только один вклад.

Система должна позволять выполнять следующие задачи:

 хранить информацию о процентах по вкладам;

 хранить информацию о клиентах;

 пополнять клиенту величину вклада;

 вычислять общую сумму выплат по процентам для всех вкладов.

Код:

Содержимое заголовочного файла:

enum Deposits {InDollars, InRubles, InEuro };

// Customer

class Customer

{

public:

Customer();

Customer(string \_fullName, double \_depositAmount, int \_validity, Deposits \_depositType);

string getFullName();

double getDepositAmount();

int getValidity();

Deposits getDepositType();

void setPercentAmount(double \_percentAmount);

double getPercentAmount();

~Customer();

private:

string fullName;

double depositAmount;

int validity;

double percentAmount;

Deposits depositType;

};

//Bank

class BankSystem

{

public:

BankSystem();

BankSystem(string \_bankName);

BankSystem(double \_dollarsRate, double \_rublesRate, double \_euroRate);

string getBankName();

double getDollarsRate();

double getRublesRate();

double getEuroRate();

void addCustomer(string \_fullName, double \_depositAmount, int \_validity, Deposits \_depositType);

double totalSumDeposits();

void customersInformation();

~BankSystem();

private:

vector<Customer> bankCustomer;

string bankName;// название учреждения

double dollarsRate, rublesRate, euroRate;

};

Содержимое основного файла:

Customer::Customer()

{

fullName = "";

depositAmount = 0.0;

validity = 0;

percentAmount = 0.0;

}

Customer::Customer(string \_fullName, double \_depositAmount, int \_validity, Deposits \_depositType)

: fullName(\_fullName), depositAmount(\_depositAmount), validity(\_validity), depositType(\_depositType)

{

percentAmount = 0.0;

}

string Customer::getFullName()

{

return fullName;

}

double Customer::getDepositAmount()

{

return depositAmount;

}

int Customer::getValidity()

{

return validity;

}

Deposits Customer::getDepositType()

{

return depositType;

}

void Customer::setPercentAmount(double \_percentAmount)

{

percentAmount = \_percentAmount;

}

double Customer::getPercentAmount()

{

return percentAmount;

}

Customer::~Customer() {}

// Bank

BankSystem::BankSystem()

{

bankName = "Priorbank";

dollarsRate = 4.7;

rublesRate = 9.4;

euroRate = 3.9;

}

BankSystem::BankSystem(string \_bankName) : bankName(\_bankName)

{

bankName = "Priorbank";

dollarsRate = 4.7;

rublesRate = 9.4;

euroRate = 3.9;

}

BankSystem::BankSystem(double \_dollarsRate, double \_rublesRate, double \_euroRate)

: dollarsRate(\_dollarsRate), rublesRate(\_rublesRate), euroRate(\_euroRate)

{

bankName = "Priorbank";

dollarsRate = 4.7;

rublesRate = 9.4;

euroRate = 3.9;

}

double BankSystem::getDollarsRate()

{

return dollarsRate;

}

double BankSystem::getRublesRate()

{

return rublesRate;

}

double BankSystem::getEuroRate()

{

return euroRate;

}

string BankSystem::getBankName()

{

return bankName;

}

void BankSystem::addCustomer(string \_fullName, double \_depositAmount, int \_validity, Deposits \_depositType)

{

bankCustomer.push\_back(Customer(\_fullName, \_depositAmount, \_validity, \_depositType));

}

void BankSystem::customersInformation()

{

double totalDollarsSum = 0;

double totalRublesSum = 0;

double totalEuroSum = 0;

for (auto &it : bankCustomer)

{

if (it.getDepositType() == 0)

{

it.setPercentAmount((it.getDepositAmount() \* dollarsRate \* static\_cast<int>(it.getValidity() \* 30.417) / 365) / 100);

cout << "Deposit in dollars" << endl;

totalDollarsSum += it.getPercentAmount();

}

else if (it.getDepositType() == 1)

{

it.setPercentAmount((it.getDepositAmount() \* rublesRate \* static\_cast<int>(it.getValidity() \* 30.417) / 365) / 100);

cout << "Deposit in rubles" << endl;

totalRublesSum += it.getPercentAmount();

}

else

{

it.setPercentAmount((it.getDepositAmount() \* euroRate \* static\_cast<int>(it.getValidity() \* 30.417) / 365) / 100);

cout << "Deposit in euro" << endl;

totalEuroSum += it.getPercentAmount();

}

cout << "Full name of the client: " << it.getFullName() << endl

<< "Deposit amount: " << it.getDepositAmount() << endl

<< "Validity: " << it.getValidity() << endl

<< "Percent amount: " << it.getPercentAmount() << endl << endl;

}

cout << "Total in dollars: " << totalDollarsSum << endl

<< "Total in rubles: " << totalRublesSum << endl

<< "Total in euro: " << totalEuroSum << endl;

}

BankSystem::~BankSystem() {}

Содержимое main файла:

BankSystem customers;

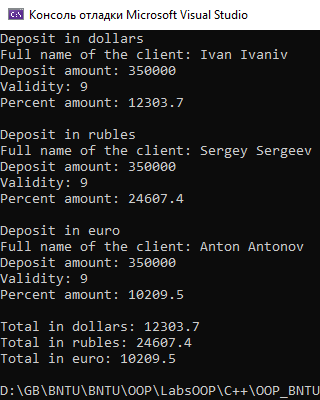
customers.addCustomer("Ivan Ivaniv", 350000, 9, InDollars);

customers.addCustomer("Sergey Sergeev", 350000, 9, InRubles);

customers.addCustomer("Anton Antonov", 350000, 9, InEuro);

customers.customersInformation();

Результат работы:



**Лабораторная работа №4**

**НАСЛЕДОВАНИЕ**

Цель работы: освоить возможности использования наследования при проектировании классов объектно-ориентированного приложения.

**Задание к работе**

1.Спроектировать иерархию, состоящую из 3–5классов.

2.Изучить механизмы явного вызова конструкторов базовых

Классов и конструкторов этого же класса с использованием ключевого слова this.

3.Использовать в программе вызов методов базового класса из методов производного при его сокрытии.

4.Использовать модификатор доступа protected.

5.Создать класс, закрытый для наследования (final), обосновать его использование.

6.Использовать ключевое слово struct.

7.Проиллюстрировать использование модификатора &.

**Индивидуальные задания**

Спроектировать иерархию для заданной предметной области:

5.Образование.

Код:

Содержимое заголовочного файла:

struct ExtraClasses

{

string name;

double price = 0.0;

};

class Education

{

public:

Education();

Education(string \_name, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers);

Education(string \_name, string \_address, string \_workTime, int \_phoneNumber);

string getName();

string getAddress();

string getWorkTime();

int getPhoneNumber();

int getNumOfStudents();

int getNumOfTeachers();

void setExtraClass(string extraClassName, double extraClassPrice);

ExtraClasses getExtraClass();

~Education();

protected:

string name, address, workTime;

int phoneNumber, numOfStudents, numOfTeachers;

ExtraClasses extraClass;

};

class School : public Education

{

public:

School();

School(string \_name, string \_depthStudy, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers);

School(string \_name, string \_address, string \_workTime, int \_phoneNumber);

string getDepthStudy();

~School();

private:

string depthStudy;

};

class Kindergarten final : public Education

{

public:

Kindergarten();

Kindergarten(string \_name, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers, int \_numOfKGTeachers);

Kindergarten(string \_name, string \_address, string \_workTime, int \_phoneNumber);

int getNumOfKGTeachers();

~Kindergarten();

private:

int numOfKGTeachers;

};

Содержимое основного файла:

Education::Education()

{

name = "";

address = "";

workTime = "";

phoneNumber = 0;

numOfStudents = 0;

numOfTeachers = 0;

this->extraClass.name = "";

this->extraClass.price = 0.0;

}

Education::Education(string \_name, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers)

: name(\_name), address(\_address), workTime(\_workTime),

phoneNumber(\_phoneNumber), numOfStudents(\_numOfStudents), numOfTeachers(\_numOfTeachers)

{

name = "";

address = "";

workTime = "";

phoneNumber = 0;

numOfStudents = 0;

numOfTeachers = 0;

this->extraClass.name = "";

this->extraClass.price = 0.0;

}

Education::Education(string \_name, string \_address, string \_workTime,

int \_phoneNumber) : name(\_name), address(\_address), workTime(\_workTime),

phoneNumber(\_phoneNumber)

{

name = "";

address = "";

workTime = "";

phoneNumber = 0;

numOfStudents = 0;

numOfTeachers = 0;

this->extraClass.name = "";

this->extraClass.price = 0.0;

}

string Education::getName()

{

return name;

}

string Education::getAddress()

{

return address;

}

string Education::getWorkTime()

{

return workTime;

}

int Education::getPhoneNumber()

{

return phoneNumber;

}

int Education::getNumOfStudents()

{

return numOfStudents;

}

int Education::getNumOfTeachers()

{

return numOfStudents;

}

void Education::setExtraClass(string extraClassName, double extraClassPrice)

{

this->extraClass.name = extraClassName;

this->extraClass.price = extraClassPrice;

}

ExtraClasses Education::getExtraClass()

{

return this->extraClass;

}

Education::~Education() {}

School::School()

{

}

School::School(string \_name, string \_depthStudy, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers)

: Education(\_name, \_address, \_workTime,\_phoneNumber, \_numOfStudents, \_numOfTeachers), depthStudy(\_depthStudy) {}

School::School(string \_name, string \_address, string \_workTime, int \_phoneNumber)

: Education(\_name, \_address, \_workTime, \_phoneNumber) {}

string School::getDepthStudy()

{

return depthStudy;

}

School::~School() {}

Kindergarten::Kindergarten()

{

numOfKGTeachers = 0;

}

Kindergarten::Kindergarten(string \_name, string \_address, string \_workTime,

int \_phoneNumber, int \_numOfStudents, int \_numOfTeachers, int \_numOfKGTeachers)

: Education(\_name, \_address, \_workTime,\_phoneNumber, \_numOfStudents, \_numOfTeachers), numOfKGTeachers(\_numOfKGTeachers)

{

numOfKGTeachers = 0;

}

Kindergarten::Kindergarten(string \_name, string \_address, string \_workTime, int \_phoneNumber)

{

numOfKGTeachers = 0;

}

int Kindergarten::getNumOfKGTeachers()

{

return numOfKGTeachers;

}

Kindergarten::~Kindergarten() {}

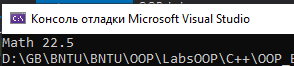
Содержимое main файла:

School school("Middle school №151", "Mathmatic", "Odintsova 25", "8:00 - 17:00", 2223344, 433, 28);

school.setExtraClass("Math", 22.5);

cout << school.getExtraClass().name << " " << school.getExtraClass().price;

Результат работы:



Класс Kindergarten (ясли-сад) закрыт для наследования так как в градации системы образования, учреждения ниже уровнем не существует.

**Лабораторная работа №5**

**ПОЛИМОРФИЗМ**

Цель работы: изучить механизмы реализации полиморфизма в C++. Ознакомиться с основными подходами при использовании абстрактных классов. Изучить шаблон проектирования Strategy.

**Задание к работе**

1.Проиллюстрировать использование абстрактных классов.

2.Применить в программе шаблон проектирования Strategy.

**Индивидуальные задания**

5.Предметная область: Банк. Система хранит информацию о вкладчиках и сделанных ими вкладах. Класс «вкладчик» содержит имя вкладчика и величину вклада. Некоторым вкладчикам при создании вклада на счет может дополнительно перечисляться фиксированная сумма. В классе «банк» реализовать методы добавления нового вкладчика и метод вычисления общей сумы вкладов**.**

Код:

Содержимое заголовочного файла:

class Contribution

{

public:

Contribution();

Contribution(double \_depositAmount);

virtual double getDepositAmount() = 0;

~Contribution();

protected:

double depositAmount;

};

class ContributionUsual : public Contribution

{

public:

ContributionUsual();

ContributionUsual(double \_depositAmount);

virtual double getDepositAmount() override;

~ContributionUsual();

};

class ContributionVip : public Contribution

{

public:

ContributionVip();

ContributionVip(double \_depositAmount, double \_bonusVip);

virtual double getDepositAmount() override;

~ContributionVip();

private:

double bonusVip;

};

class Contributer

{

public:

Contributer();

Contributer(string \_fullName, Contribution\* \_contribution);

string getFullName();

void setStrategy(Contribution\* \_contribution);

virtual double getDepositAmount();

~Contributer();

private:

string fullName;

Contribution\* contribution;

};

class Bank2

{

public:

Bank2();

void addContributers(Contributer\* \_contributers);

void information();

~Bank2();

private:

vector<Contributer\*> contributers;

};

Содержимое основного файла:

Contribution::Contribution() {}

Contribution::Contribution(double \_depositAmount) : depositAmount(\_depositAmount) {}

Contribution::~Contribution() {}

ContributionUsual::ContributionUsual(){}

ContributionUsual::ContributionUsual(double \_depositAmount) : Contribution(\_depositAmount) {}

double ContributionUsual::getDepositAmount()

{

return depositAmount;

}

ContributionUsual::~ContributionUsual() {}

ContributionVip::ContributionVip()

{

bonusVip = 0.0;

}

ContributionVip::ContributionVip(double \_depositAmount, double \_bonusVip) : Contribution(\_depositAmount), bonusVip(\_bonusVip) {}

double ContributionVip::getDepositAmount()

{

return depositAmount + bonusVip;

}

ContributionVip::~ContributionVip() {}

Contributer::Contributer() {}

Contributer::Contributer(string \_fullName, Contribution\* \_contribution) : fullName(\_fullName), contribution(\_contribution) {}

string Contributer::getFullName()

{

return fullName;

}

void Contributer::setStrategy(Contribution\* \_contribution)

{

if (contribution != nullptr)

{

delete contribution;

}

contribution = \_contribution;

}

double Contributer::getDepositAmount()

{

return contribution->getDepositAmount();

}

Contributer::~Contributer()

{

delete contribution;

}

Bank2::Bank2() {}

void Bank2::addContributers(Contributer\* \_contributers)

{

contributers.push\_back(\_contributers);

}

void Bank2::information()

{

double sum = 0;

for (auto it : contributers)

{

cout << "Contributer: " << it->getFullName() << ", contribution: " << it->getDepositAmount() << endl;

sum += it->getDepositAmount();

}

cout << "Number of contributers: " << contributers.size() << endl;

cout << "Contributions amount: " << sum << endl;

}

Bank2::~Bank2() {}

Содержимое main файла:

Contributer\* contributer1 = new Contributer("Vitold Ivanov", new ContributionVip(3000, 40));

contributer1->setStrategy(new ContributionUsual( 2500));

Bank2 bank;

bank.addContributers(contributer1);

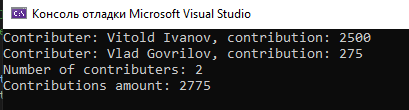
Contributer\* contributer2 = new Contributer("Vlad Govrilov", new ContributionUsual(300));

contributer2->setStrategy(new ContributionVip(250, 25));

bank.addContributers(contributer2);

bank.information();

Результат работы:



**Лабораторная работа №6**

Построить иерархию классов – **персона, студент, преподаватель**. В базовом классе реализовать виртуальный метод. В производных классах переопределить виртуальный метод. В функции Main() проверить работоспособность созданной иерархии классов и переопределенных методов.

Код:

Содержимое заголовочного файла:

class Person

{

public:

Person();

Person(string \_name, int \_age);

string getName();

virtual int getAge() = 0;

~Person();

protected:

string name;

int age;

};

class Student : public Person

{

public:

Student();

Student(string \_name, int \_age);

virtual int getAge() override;

~Student();

};

class Teacher : public Person

{

public:

Teacher();

Teacher(string \_name, int \_age);

virtual int getAge() override;

~Teacher();

};

Содержимое основного файла:

Person::Person() {}

Person::Person(string \_name, int \_age) : name(\_name), age( \_age) {}

string Person::getName()

{

return name;

}

Person::~Person(){}

Student::Student() {}

Student::Student(string \_name, int \_age) : Person(\_name, \_age) {}

int Student::getAge()

{

return age;

}

Student::~Student() {}

Teacher::Teacher() {}

Teacher::Teacher(string \_name, int \_age) : Person(\_name, \_age) {}

int Teacher::getAge()

{

return age;

}

Teacher::~Teacher() {}

Содержимое main файла:

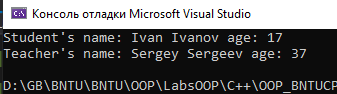
Student student("Ivan Ivanov", 17);

cout << "Student's name: " << student.getName() << " age: " << student.getAge() << endl;

Teacher teacher("Sergey Sergeev", 37);

cout << "Teacher's name: " << teacher.getName() << " age: " << teacher.getAge() << endl;

Результат работы:



**Лабораторная работа №7**

**Задача №5.**

Подсчитать, сколько раз каждое слово встречается в заданном тексте. Результат записать в коллекцию **Map <TKey, TValue>**. Текс использовать из приложения 1. Вывести статистику по тексту в консоль.

**Приложение 1.**

Вот дом, Который построил Джек. А это пшеница, Которая в темном чулане хранится В доме, Который построил Джек. А это веселая птица-синица, Которая часто ворует пшеницу, Которая в темном чулане хранится В доме, Который построил Джек.

Содержимое main файла:

const int N = 256;

char line[N];

cin.getline(line, N);

map<string, int> dictionary;

#pragma warning(disable : 4996)

for (char\* p = strtok(line, " -.,/!?"); p != nullptr; p = strtok(nullptr, " -.,/!?"))

{

string word(p);

auto it = dictionary.find(word);

if (it != dictionary.end())

{

++it->second;

}

else

{

dictionary.emplace\_hint(it, move(word), 1);

}

}

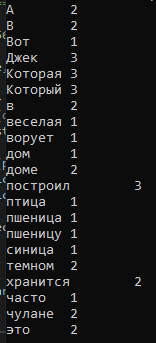
for (auto& iter : dictionary)

{

cout << iter.first << "\t" << iter.second << '\n';

}

Результат работы:



# Лабораторная работа 8. Разработка классов.

## Цель работы. Получить навыки проектирования простейших классов. Научиться создавать объекты класса. Освоить принцип инкапсуляции.

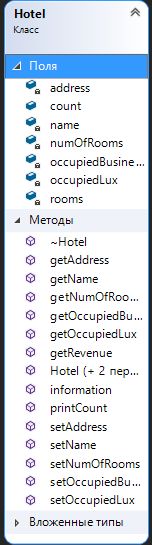
## Задание.

* Спроектировать класс согласно варианту индивидуального задания. Для класса использовать отдельный модуль.
* Предусмотреть у класса наличие полей, методов и свойств. **Названия членов класса должны быть осмысленны и снабжены комментариями**.
* Для всех нетекстовых полей разработать свойства, контролирующие корректность значения (например, оценка – от 0 до 10). Одно из свойств (любое) должно быть только для записи, другое – только для чтения с инициализацией в конструкторе. Поля, связанные со свойствами должны быть инкапсулированы.
* Разработать метод для отображения всех полей класса.
* Добавить статическое поле – количество всех созданных объектов вашего класса. Разработать метод, который выводит на экран это поле.
* Разработать несколько конструкторов класса (не менее 3). Один – без аргументов, с заполнением полей значением по умолчанию, второй – с полным набором аргументов и третий с частичным набором аргументов. Связать конструкторы в цепочку, дабы избежать дублирования кода. Для проверки корректности аргументов в конструкторах использовать свойство.
* Отобразить диаграмму спроектированного класса (автоматическую, в развернутом виде).
* Создать несколько объектов класса. Продемонстрировать использование всех конструкторов.
* Продемонстрировать работу всех методов.
* Продемонстрировать использование всех свойств – для получения и присваивания значения.
* Создать еще одну переменную, присвоив ей один из предыдущих объектов. Продемонстрировать, что две переменные ссылаются на один объект.

**Индивидуальные задания**

5.Класс «Гостиница». Хранить информацию о названии и адресе гостиницы, количестве номеров (есть два класса номеров – бизнес и люкс), количестве занятых номеров каждого класса, стоимости каждого вида номера (за сутки). Реализовать метод для определения выручки гостиницы за день.

Диаграмма класса:



Код:

Содержимое заголовочного файла:

class Hotel

{

public:

Hotel();// конструктор по умолчанию

Hotel(string \_name, int \_numOfRooms);

Hotel(string \_name, string \_address, int \_numOfRooms, int \_occupiedBusiness, int \_occupiedLux);// пользовательский конструктор

static int count;

void setName(string \_name);//set-функция для поля name

void setAddress(string \_address);//set-функция для поля address

void setNumOfRooms(int \_numOfRooms);//set-функция для поля numOfRooms

void setOccupiedBusiness(int \_occupiedBusiness);//set-функция для поля occupiedBusiness

void setOccupiedLux(int \_occupiedLux);//set-функция для поля occupiedLux

string getName();//get-функция для поля name

string getAddress();//get-функция для поля address

int getNumOfRooms();//get-функция для поля numOfRooms

int getOccupiedBusiness();//get-функция для поля occupiedBusiness

int getOccupiedLux();//get-функция для поля occupiedLux

double getRevenue();//get-функция для подсчета общей выручки

void printCount();

void information();

~Hotel();// деструктор

private:

struct Rooms// структура с полями типов гостинечных номеров и их стоимость

{

double Business = 18.3;

double Lux = 24.4;

};

string name, address;// поля класса

int numOfRooms, occupiedBusiness, occupiedLux;// поля класса

Rooms rooms;// поля класса

};

Содержимое основного файла:

int Hotel::count = 0;

Hotel::Hotel()

{

name = "";

address = "";

numOfRooms = 0;

occupiedBusiness = 0;

occupiedLux = 0;

count++;

}

Hotel::Hotel(string \_name, int \_numOfRooms)

{

setName(\_name);

setNumOfRooms(\_numOfRooms);

}

Hotel::Hotel(string \_name, string \_address, int \_numOfRooms, int \_occupiedBusiness, int \_occupiedLux)

{

setName(\_name);

setAddress(\_address);

setNumOfRooms(\_numOfRooms);

setOccupiedBusiness(\_occupiedBusiness);

setOccupiedLux(\_occupiedLux);

count++;

}

void Hotel::setName(string \_name)

{

name = \_name;

}

void Hotel::setAddress(string \_address)

{

address = \_address;

}

void Hotel::setNumOfRooms(int \_numOfRooms)

{

if (\_numOfRooms < 0)

{

\_numOfRooms = 0;

}

numOfRooms = \_numOfRooms;

}

void Hotel::setOccupiedBusiness(int \_occupiedBusiness)

{

if (\_occupiedBusiness < 0)

{

\_occupiedBusiness = 0;

}

occupiedBusiness = \_occupiedBusiness;

}

void Hotel::setOccupiedLux(int \_occupiedLux)

{

if (\_occupiedLux < 0)

{

\_occupiedLux = 0;

}

occupiedLux = \_occupiedLux;

}

string Hotel::getName()

{

return name;

}

string Hotel::getAddress()

{

return address;

}

int Hotel::getNumOfRooms()

{

return numOfRooms;

}

int Hotel::getOccupiedBusiness()

{

return occupiedBusiness;

}

int Hotel::getOccupiedLux()

{

return occupiedLux;

}

double Hotel::getRevenue()

{

if ((occupiedBusiness + occupiedLux) > numOfRooms)

{

return 0;

}

return (static\_cast<double>(occupiedBusiness) \* rooms.Business) + (static\_cast<double>(occupiedLux) \* rooms.Lux);

}

void Hotel::printCount()

{

cout << "Number of created classes: " << count << endl;

}

void Hotel::information()

{

if ((getOccupiedBusiness() + getOccupiedLux()) > getNumOfRooms())

{

cout << "Name of hotel: " << getName() << endl

<< "Address: " << getAddress() << endl

<< "Error" << endl;

}

else

{

cout << "Name of hotel: " << getName() << endl

<< "Address: " << getAddress() << endl

<< "Number of rooms: " << getNumOfRooms() << endl

<< "Occupied Business rooms: " << getOccupiedBusiness() << endl

<< "Occupied Lux rooms: " << getOccupiedLux() << endl

<< "Revenue: " << getRevenue() << endl;

}

}

Hotel::~Hotel() {}

Содержимое main файла:

Hotel hotel("Viktoria", "Masherova 105", 10, 6, 4);

hotel.information();

hotel.printCount();

cout << endl;

Hotel hotel2("Hilton", "Orlovskaya 23", 11, 3, 5);

hotel2.information();

hotel2.printCount();

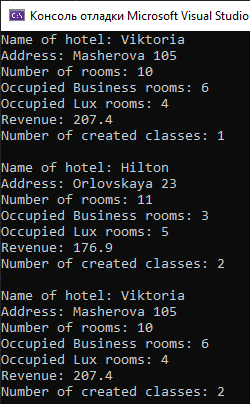
cout << endl;

Hotel hotel3 = hotel;

hotel3.information();

hotel3.printCount();

Результат работы:



# Лабораторная работа 9. Ассоциации.

## Цель работы. Изучить варианты ассоциаций классов. Освоить возможности членов класса с модификатором static.

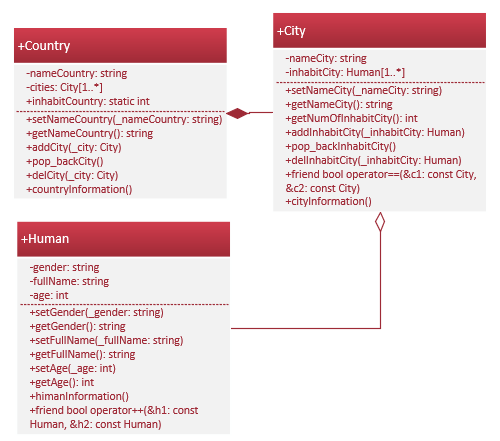
## Задание.

Спроектировать UML-диаграммы классов согласно варианту индивидуального задания. Эта диаграмма создается вручную в разделе Architecture в MS Visual Studio или в MS Visio или в любом другом подходящем редакторе. Эта диаграмма является вашим проектом, на основе которого вы создаете классы.

В каждом классе предусмотреть поля, свойства и методы.

Между парами классов спроектировать отношения ­– композицию и агрегацию.

Диаграмма:



Код:

Содержимое заголовочного файла:

class Human

{

public:

Human();

Human(string \_gender, string \_fullName, int \_age);

void setGender(string \_gender);

string getGender();

void setFullName(string \_fullName);

string getFullName();

void setAge(int \_age);

int getAge();

void humanInformation();

friend bool operator==(const Human &h1, const Human &h2);

~Human();

private:

string gender, fullName;

int age;

};

class Country

{

public:

Country();

Country(string \_nameCountry);

class City

{

public:

City();

City(string \_nameCity);

City(string \_nameCity, Human \_inhabitCity);

void setNameCity(string \_nameCity);

string getNameCity();

int getNumOfInhabitCity();

void addInhabitCity(Human \_inhabitCity);

void pop\_backInhabitCity();

void delInhabitCity(Human \_inhabitCity);

friend bool operator==(const City& c1, const City& c2);

void cityInformation();

~City();

private:

string nameCity;

vector <Human> inhabitCity;

};

static int inhabitCountry;

void setNameCountry(string \_nameCountry);

string getNameCountry();

void addCity(City \_city);

void pop\_backCity();

void delCity(City \_cities);

void countryInformation();

~Country();

private:

vector<City> cities;

string nameCountry;

};

Содержимое основного файла:

int Country::inhabitCountry = 0;

Country::Country()

{

nameCountry = "";

}

Country::Country(string \_nameCountry)

{

setNameCountry(\_nameCountry);

}

void Country::setNameCountry(string \_nameCountry)

{

nameCountry = \_nameCountry;

}

string Country::getNameCountry()

{

return nameCountry;

}

void Country::addCity(City \_city)

{

cities.push\_back(\_city);

}

void Country::pop\_backCity()

{

cities.pop\_back();

}

void Country::delCity(City \_cities)

{

inhabitCountry -= \_cities.getNumOfInhabitCity();

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

{

if (cities[i] == \_cities)

{

cities[i] = move(cities.back());

pop\_backCity();

}

}

}

void Country::countryInformation()

{

cout << "The name of the country: " << getNameCountry() << endl

<< "Number of inhabitants: " << inhabitCountry << endl

<< "Cities: " << endl;

for (auto& it : cities)

{

cout << it.getNameCity() << endl;

}

cout << endl;

}

Country::~Country() {}

Country::City::City()

{

nameCity = "";

}

Country::City::City(string \_nameCity)

{

setNameCity(\_nameCity);

}

Country::City::City(string \_nameCity, Human \_inhabitCity)

{

setNameCity(\_nameCity);

addInhabitCity(\_inhabitCity);

}

void Country::City::setNameCity(string \_nameCity)

{

nameCity = \_nameCity;

}

string Country::City::getNameCity()

{

return nameCity;

}

int Country::City::getNumOfInhabitCity()

{

return inhabitCity.size();

}

void Country::City::addInhabitCity(Human \_inhabitCity)

{

inhabitCountry++;

inhabitCity.push\_back(\_inhabitCity);

}

void Country::City::pop\_backInhabitCity()

{

inhabitCountry--;

inhabitCity.pop\_back();

}

void Country::City::delInhabitCity(Human \_inhabitCity)

{

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

{

if (inhabitCity[i] == \_inhabitCity)

{

inhabitCity[i] = move(inhabitCity.back());

pop\_backInhabitCity();

}

}

}

bool operator==(const Country::City& c1, const Country::City& c2)

{

return(c1.nameCity == c2.nameCity);

}

void Country::City::cityInformation()

{

cout << "City name: " << getNameCity() << endl

<< "Number of inhabitants: " << getNumOfInhabitCity() << endl

<< "Inhabitants:" << endl << endl;

for(auto &it : inhabitCity)

{

cout << "Gender: " << it.getGender() << endl

<< "Full Name: " << it.getFullName() << endl

<< "Age: " << it.getAge() << endl << endl;

}

}

Country::City::~City() {}

Human::Human()

{

gender = "";

fullName = "";

age = 0;

}

Human::Human(string \_gender, string \_fullName, int \_age)

{

setGender(\_gender);

setFullName(\_fullName);

setAge(\_age);

}

void Human::setGender(string \_gender)

{

gender = \_gender;

}

string Human::getGender()

{

return gender;

}

void Human::setFullName(string \_fullName)

{

fullName = \_fullName;

}

string Human::getFullName()

{

return fullName;

}

void Human::setAge(int \_age)

{

age = \_age;

}

int Human::getAge()

{

return age;

}

bool operator==(const Human& h1, const Human& h2)

{

return (h1.age == h2.age && h1.fullName == h2.fullName && h1.gender == h2.gender);

}

void Human::humanInformation()

{

cout << endl;

cout << "Gender: " << getGender() << endl

<< "Full Name: " << getFullName() << endl

<< "Age: " << getAge() << endl;

}

Human::~Human() {}

Содержимое main файла:

Country Belarus("Belarus");

Human Vasiliy("male", "Vasily", 30);

Human Fedor("male", "Fedor", 16);

Human Alice("female", "Alice", 22);

Human Victoria("female", "Victoria", 25);

Country::City Minsk("Minsk", Vasiliy);

Country::City Gomel("Gomel", Alice);

Country::City Vitebsk("Vitebsk");

Country::City Brest("Brest");

Belarus.addCity(Vitebsk);

Belarus.addCity(Brest);

Belarus.addCity(Gomel);

Belarus.addCity(Minsk);

Gomel.addInhabitCity(Victoria);

Minsk.addInhabitCity(Fedor);

Minsk.cityInformation();

Gomel.cityInformation();

Belarus.countryInformation();

Minsk.delInhabitCity(Vasiliy);

Minsk.cityInformation();

Belarus.delCity(Vitebsk);

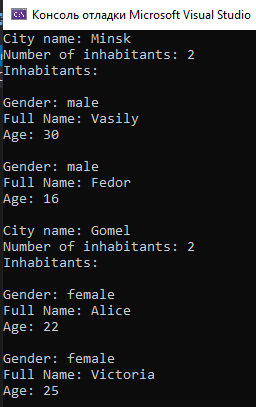
Belarus.delCity(Minsk);

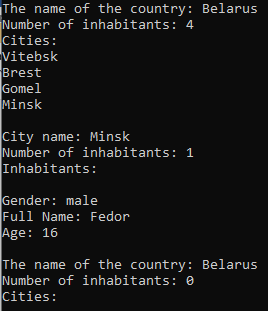
Belarus.delCity(Brest);

Belarus.delCity(Gomel);

Belarus.countryInformation();

Результат работы:





# ЗАКЛЮЧЕНИЕ

ООП является способом организации программы. Основное внимание при его изучении уделяется организации программы, а не вопросам написания кода. Главным компонентом объектно-ориентированной программы является объект, содержащий данные и функции для их обработки. Класс является формой или образцом для множества сходных между собой объектов.

В С++ существует 3 вида циклов. Цикл for чаще всего используется в тех случаях, когда число исполнений цикла известно заранее. Циклы while и do используются тогда, когда условие для завершения цикла формируется в процессе выполнения цикла, причем тело цикла while может не исполняться ни разу, а тело цикла do всегда исполняется хотя бы один раз. Тело цикла может представлять собой как один оператор, так и блок операторов, заключенный в фигурные скобки. Переменная, определенная внутри блока, доступна, или видима, только внутри этого блока.

Структура является важным аспектом С++, поскольку её синтаксис совпадает с синтаксисом класса. Классы представляют собой не что иное, как структуры, включающие в себя функции. Типичной целью использования структур является объединение данных различных типов в один программный объект.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Язык программирования C++. Краткий курс, 2019. – 156 с.

2. Орленко, Евдокимов C++ на примерах. Практика, практика и только практика, 2009. – 274 с.

3. Марапулец Ю. Язык С++. Основы программирования, 2010. – 184 с.

4. Культин Н. CC++ в задачах и примерах, 2004. – 584 с.

5. Р.Лафоре Объектно-ориентированное программирование в С++, 2004. – 768 с.