Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №2**

По дисциплине: “Языки программирования”

Тема: «Наследование и виртуальные функции»**Вариант №10**

**Выполнил**:

студент 2 курса

группы ПО-7

Курмыса Е.Е.

**Проверила:**

Дряпко А. В.

Брест 2021

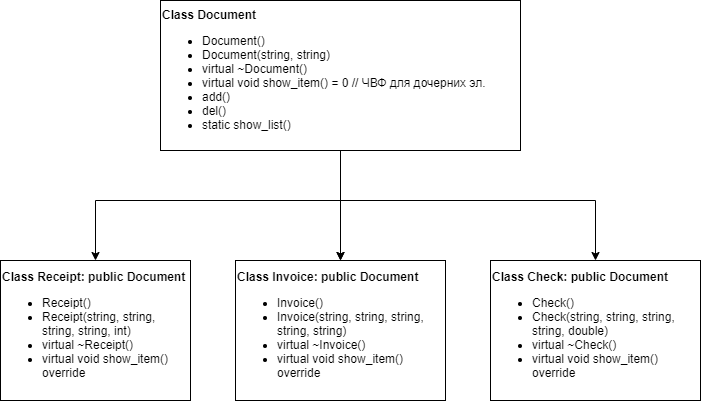
**Цель работы**: получить практические навыки создания иерархии классов и использования статических компонентов класса в C++.

**Постановка задачи**

Требуется написать программу, в которой создаётся иерархия классов. Включить полиморфные объекты в связанный список, используя статические элементы класса. Показать использование виртуальных функций.

Классы: квитанция (Receipt), накладная (Invoice), документ (Document), чек (Check). Легко заметить, что для них желательно создать конструкторы (с параметрами и без) + деструкторы с той целью, чтобы наглядно, при помощи команд cout, показать на консоли создание/удаление объекта.

Очевидно, что квитанции, накладные и чеки являются типами документов, так что мы имеем единственный родительский класс Document, от которого наследуются 3 остальные клас са со своими функциями:



Как видно из диаграммы выше, мы определяем чистый виртуальный метод show\_item() в базовом классе Document — он будет использоваться ТОЛЬКО для объектов дочерних классов и для каждого из них он выглядит по-своему.

Далее, у нас должны иметься статические переменные, которые будут подсчитывать как количество всех документов (document\_count), так количества определённых типов документов. Также мы определим статический список, в котором будут храниться элементы при создании/явном вызове функции add(), а также количество элементов в нём (index — порядковый номер последнего элемента в списке).

**Программный код (написан на C++):**

**classes.h (определения классов и их методов + статических переменных):**

#ifndef CLASSES\_H

#define CLASSES\_H

using namespace std;

class Document { // класс Документ

public:

Document();

Document(string, string);

virtual ~Document(); // так как есть виртуальные функции, то и деструктор должен быть виртуальным

virtual void show\_item() = 0; // чистая виртуальная функция для дочерних классов

void add(); // функция добавления элементов в список

void del(); // удаление элементов из списка (при деструкторе)

static void show\_list(); // для вывода списка элементов

protected:

string date; // дата создания

string organisation; // организация, указанная в документе

static int index; // порядковый номер документа в списке

static int document\_count; // статический элемент - кол-во документов

static Document \*\*documents; // список документов

};

class Receipt: public Document { // класс Квитанция

public:

Receipt();

Receipt(string, string, string, string, int);

~Receipt();

void show\_item() override; // перезаписываем ЧВ-функцию из родительского в дочерний класс

protected:

string sender; // отправитель денег или иных ценностей

string receiver; // получатель денег или иных ценностей

int cost; // стоимость денег или иных ценностей

static int receipt\_count; // кол-во квитанций

};

class Invoice: public Document { // класс Накладная

public:

Invoice();

Invoice(string, string, string, string, string);

~Invoice();

void show\_item() override;

protected:

string goods; // перевозимый товар, его описание

string provider; // поставщик товара

string date\_of\_delivery; // дата доставки товара

static int invoice\_count; // кол-во накладных

};

class Check: public Document { // Класс Чек

public:

Check();

Check(string, string, string, string, double);

~Check();

void show\_item() override;

protected:

string payee; // чекодержатель

string drawer; // чекодатель

double amount; // цена, указанная в чеке

static int check\_count;

};

#endif

**classes.cpp (объявление функций для каждого из классов):**

#include <iostream>

#include <cstring>

#include "classes.h"

using namespace std;

// конструктор по умолчанию для класса Document

Document::Document() {

date = "1 января 1970";

organisation = "Feel Good Inc.";

add();

cout << "Создан объект класса Document номер " << ++document\_count << " через конструктор по умолчанию.\n";

}

// конструктор с параметрами для класса Document

Document::Document(string date, string organisation):

date(date), organisation(organisation) {

add();

cout << "Создан объект класса Document номер " << ++document\_count << " через конструктор с параметрами.\n";

}

// деструктор для класса Document

Document::~Document() {

del();

cout << "Уничтожен объект класса Document номер " << document\_count-- << ".\n";

}

// добавление нового элемента в список (при конструкторе или явном вызове)

void

Document::add() {

Document \*\*temp = documents;

documents = new Document\* [index + 1];

for (int i = 0; i < index; i++) documents[i] = temp[i];

documents[index] = this;

index++;

}

// удаление последнего элемента из списка (при деструкторе)

void

Document::del() {

Document \*\*temp = documents;

documents = new Document\* [index];

for (int i = 0; i < index - 1; i++) documents[i] = temp[i];

index--;

}

// показ списка элементов документов

void

Document::show\_list() {

cout << "\nСписок всех документов:\n";

for (int i = 0; i < index; i++) documents[i]->show\_item();

}

// конструктор по умолчанию для класса Receipt

Receipt::Receipt():

Document("1 января 1970 г.", "Feel Good Inc.") {

sender = "Иванов И.И.";

receiver = "Сидоров С.С.";

cost = 150;

cout << "Создан объект класса Receipt номер " << ++receipt\_count << " через конструктор по умолчанию.\n";

}

// конструктор с параметрами для класса Receipt

Receipt::Receipt(string date, string organisation, string sender, string receiver, int cost):

Document(date, organisation), sender(sender), receiver(receiver), cost(cost) {

cout << "Создан объект класса Receipt номер " << ++receipt\_count << " через конструктор с параметрами.\n";

}

// деструктор для класса Receipt

Receipt::~Receipt() {

cout << "Уничтожен объект класса Receipt номер " << receipt\_count-- << ".\n";

}

// вывод информации для объекта класса Receipt

void

Receipt::show\_item() {

cout << "Квитанция:" << endl;

cout << "Дата: " << this->date << endl;

cout << "Организация: " << this->organisation << endl;

cout << "Отправитель: " << this->sender << endl;

cout << "Получитель: " << this->receiver << endl;

cout << "Стоимость: " << this->cost << endl << endl;

}

// конструктор по умолчанию для класса Invoice

Invoice::Invoice():

Document("1 января 1970 г.", "Feel Good Inc.") {

goods = "Мясо, картошка, киви, репчатый лук, оливье";

provider = "Маменко И.В.";

date\_of\_delivery = "8 сентября 2008 г.";

cout << "Создан объект класса Invoice номер " << ++invoice\_count << " через конструктор по умолчанию.\n";

}

// конструктор с параметрами для класса Invoice

Invoice::Invoice(string date, string organisation, string goods, string provider, string date\_of\_delivery):

Document(date, organisation), goods(goods), provider(provider), date\_of\_delivery(date\_of\_delivery) {

cout << "Создан объект клааса Invoice номер " << ++invoice\_count << " через конструктор с параметрами.\n";

}

// деструктор для класса Invoice

Invoice::~Invoice() {

cout << "Уничтожен объект класса Invoice номер " << invoice\_count-- << ".\n";

}

// вывод информации для объекта класса Invoice

void

Invoice::show\_item() {

cout << "Накладная:" << endl;

cout << "Дата: " << this->date << endl;

cout << "Огранизация: " << this->organisation << endl;

cout << "Товары: " << this->goods << endl;

cout << "Поставщик: " << this->provider << endl;

cout << "Дата поставки: " << this->date\_of\_delivery << endl << endl;

}

// конструктор по умолчанию для класса Check

Check::Check():

Document("1 января 1970 г.", "Feel Good Inc.") {

payee = "Максимова О.Г.";

drawer = "Дементьева Т.Г.";

amount = 288.20;

cout << "Создан объект класса Check номер " << ++check\_count << " через конструктор по умолчанию.\n";

}

// конструктор с параметрами для класса Check

Check::Check(string date, string organisation, string payee, string drawer, double amount):

Document(date, organisation), payee(payee), drawer(drawer), amount(amount) {

cout << "Создан объект класса Check номер " << ++check\_count << " через конструктор с параметрами.\n";

}

// деструктор для класса Check

Check::~Check() {

cout << "Унчитожен объект класса Check номер " << check\_count-- << ".\n";

}

// вывод информации для объекта класса Check

void

Check::show\_item() {

cout << "Чек:" << endl;

cout << "Дата: " << this->date << endl;

cout << "Огранизация: " << this->organisation << endl;

cout << "Чекодержтель: " << this->payee << endl;

cout << "Чекодатель: " << this->drawer << endl;

cout << "Сумма (в бел. руб.): " << this->amount << endl << endl;

}

**main.cpp (основная функция main(), в которой исполняются требуемые функции, а также объявление статических переменных):**

#include <iostream>

#include "classes.h"

int Document::document\_count = 0;

int Document::index = 0;

int Receipt::receipt\_count = 0;

int Invoice::invoice\_count = 0;

int Check::check\_count = 0;

Document\*\* Document::documents = nullptr;

int main() {

// неявное добавление объектов в список (через конструктор)

Receipt doc1;

Receipt doc2("1 ноября 2001 г.", "Vitruvium", "Гордеев А.Г.", "Филатова В.А.", 846);

Invoice doc3;

Invoice doc4("29 февраля 2004 г.", "Карацинов и компания", "Трубопрокатные материалы", "Карацинов А.В.", "7 марта 2004 г.");

Check doc5;

Check doc6("15 сентября 2021 г.", "Белагропромбанк", "Коновалова С.В.", "Мальцева Е.Д.", 500.39);

// явное добавление объектов в список путём вызова add()

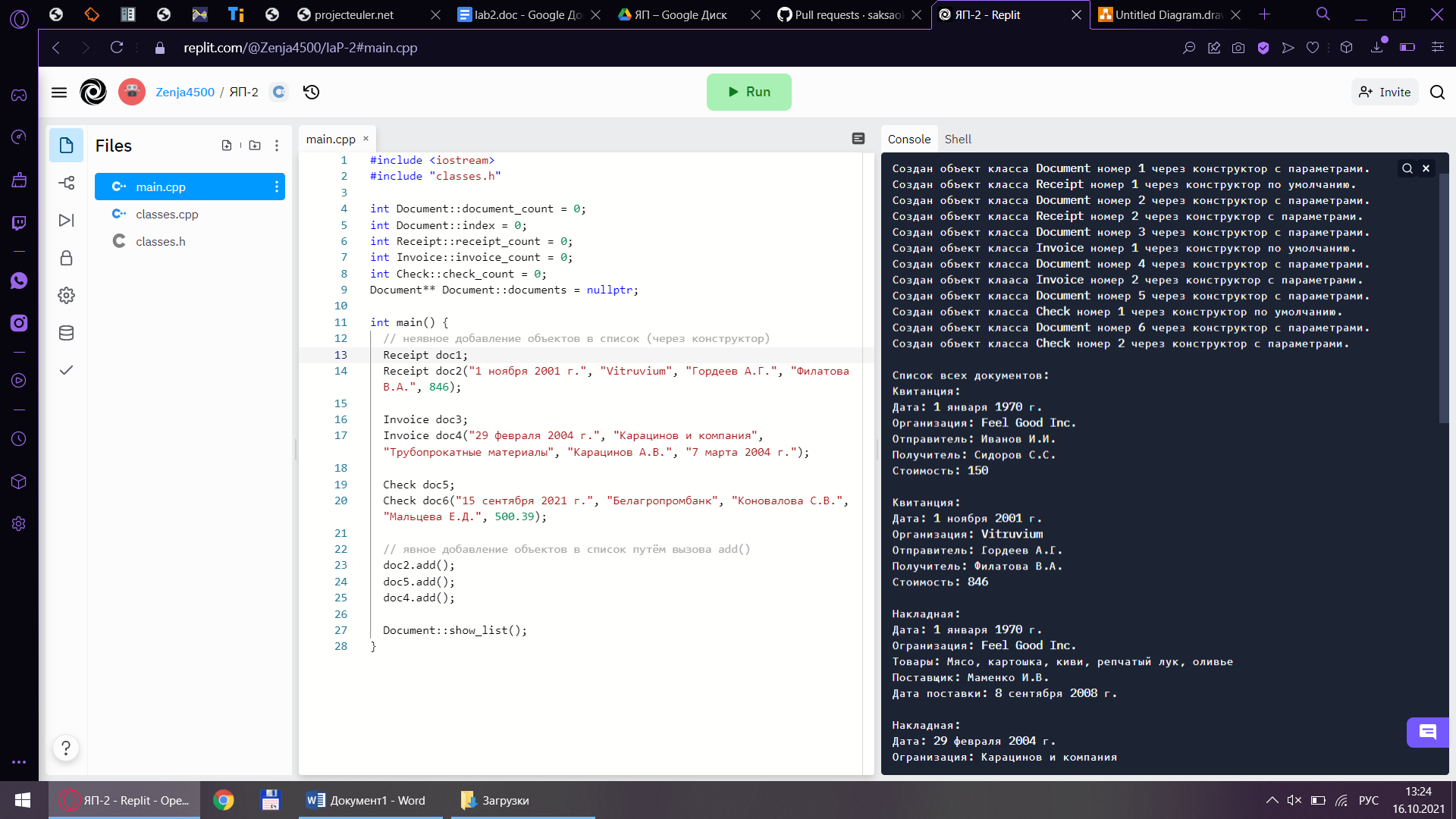
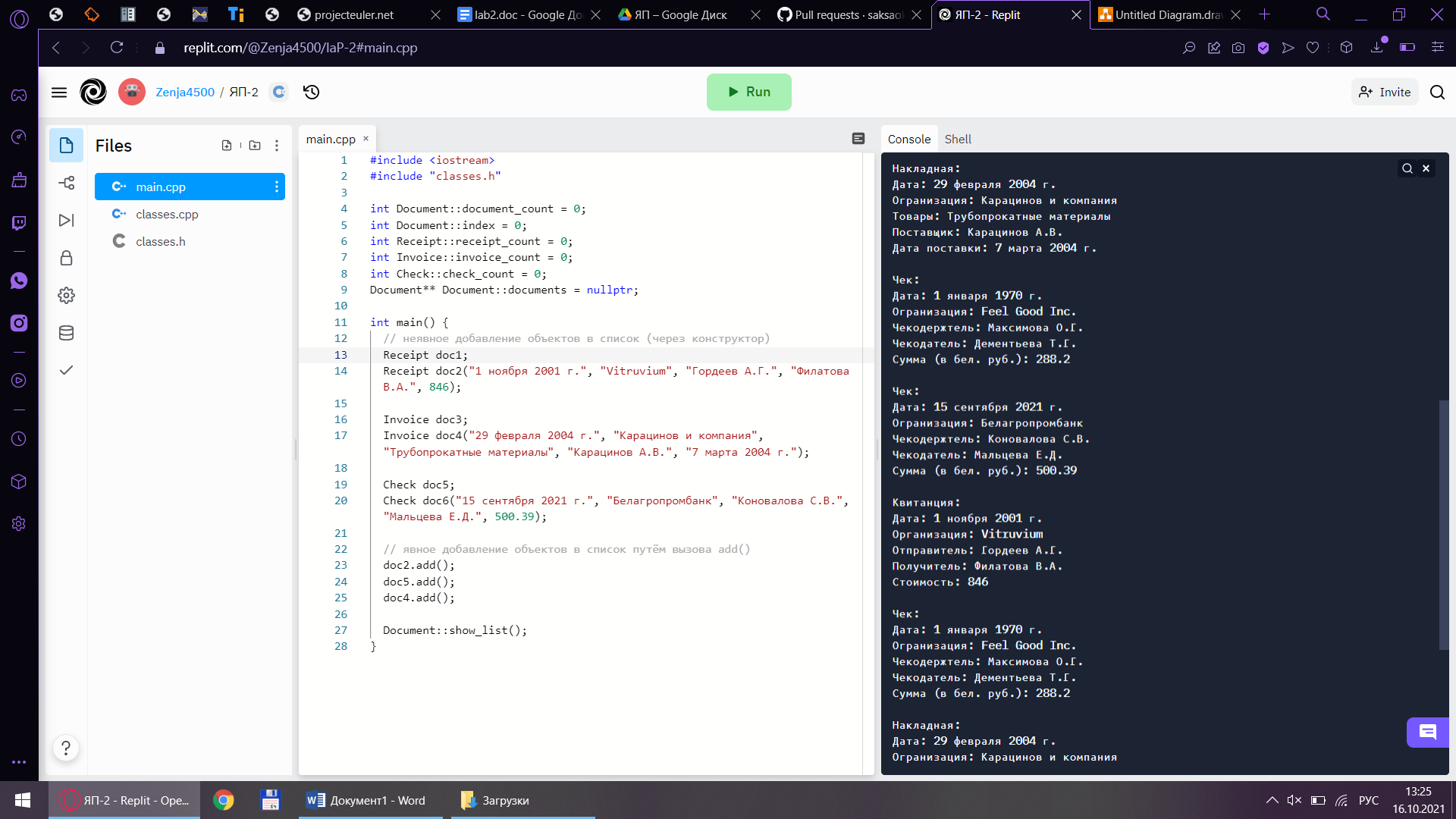
doc2.add();

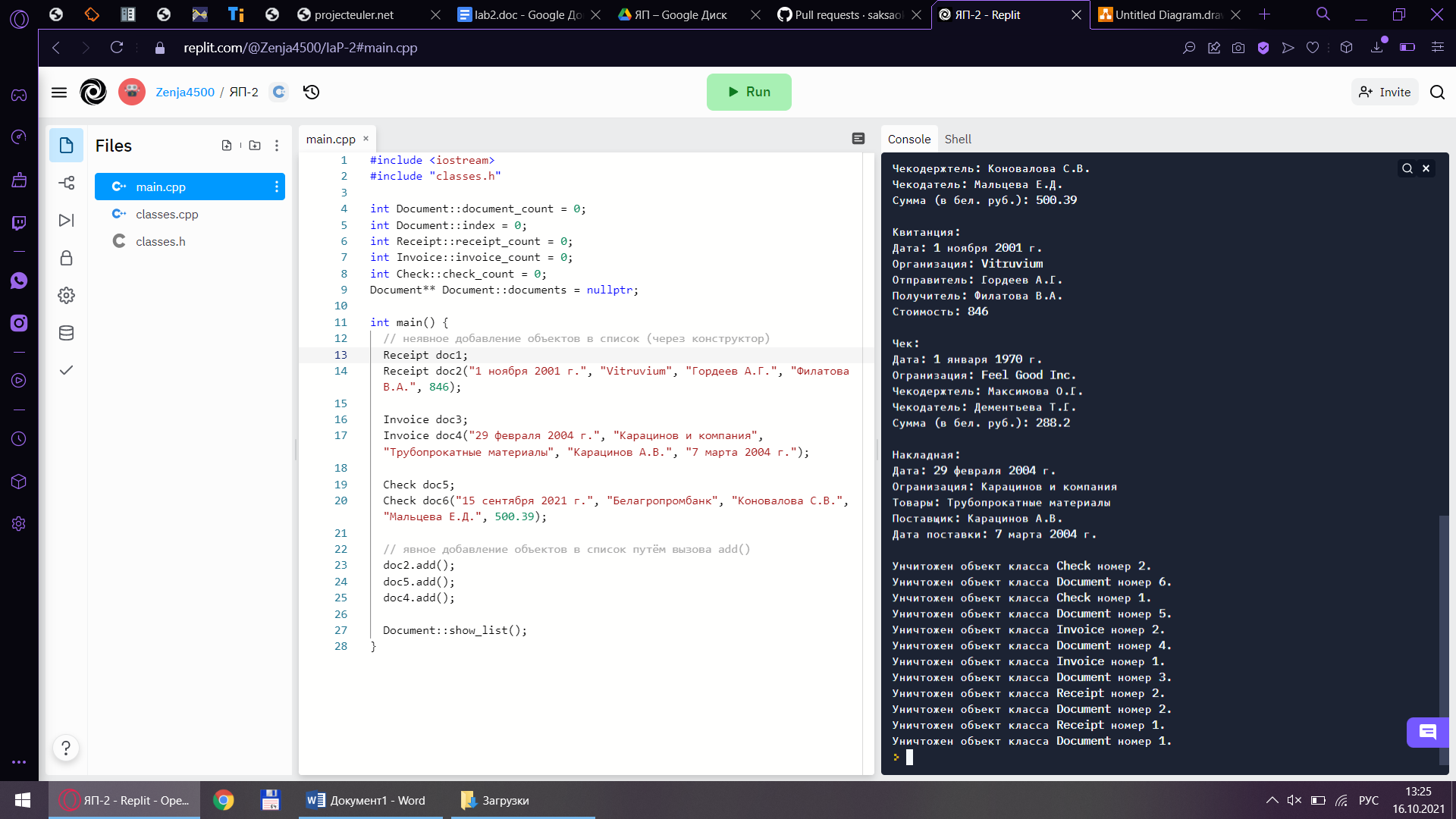
doc5.add();

doc4.add();

Document::show\_list();

}



Стоит показать необходимость виртуальных функций в программе. Во-первых, необходимость виртуальности деструкторов каждого из классов обусловлена тем, что в них присутствует хотя бы один виртуальный метод, следовательно, и деструктор тоже должен быть виртуальным.

Во-вторых, поскольку мы должны показывать содержимое каждого элемента списка через show\_list(), проходя через каждый объект списка в цикле и вызывая для него show\_item(), то тогда нужно сделать таким образом, чтобы функция вызова была определена для всех классов, но описана при этом только в дочерних классах (в родительском это делать необязательно).

В таком случае в родительском классе стоит создать сначала чистую виртуальную функцию show\_item(), которая доступна как для объектов класса Document, так и для объектов дочерних классов, а затем создать такие же функции в дочерних классах, предварительно переопределив их с помощью override. Таким образом мы сможем использовать одну и ту же функцию в разных классах и сделать так, чтобы она выводила разные данные в зависимости от класса объекта.

Если же мы не будем использовать виртуальные функции, то тогда получится так, что все функции будут формально независимы друг от друга. Однако, поскольку таких функций будет несколько, то при её вызове компилятор не будет знать, какую именно функцию использовать, и выдаст ошибку компиляции программы. Таким образом, лучше всего сделать «дерево» виртуальных функций, чтобы они были связаны между собой и при этом они могли бы однозначно исполняться в зависимости от того, какого класса тот или иной объект.

**Вывод:** я получил практические навыки создания иерархии классов и использования статических компонентов класса в C++.