Лабораторная работа 2.

Объекты и классы (наследование, конструкторы, деструкторы)

Цель работы:

Продолжить формирование знаний и умений, необходимых для создания объектов, классов и механизмом наследования в C#.

Теоретические сведения (повторение)

**Классы и объекты**

Класс является основой для создания объектов. В классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса.

**Методы и переменные**, составляющие класс, называются членами класса. При определении класса объявляются данные, которые он содержит, и код, работающий с этими данными. Данные содержатся в переменных экземпляра, которые определены классом, а код содержится в методах. В С# определены несколько специфических разновидностей членов класса. Это — переменные экземпляра, статические переменные, константы, методы, конструкторы, деструкторы, индексаторы, события, операторы и свойства.

Непосредственно инициализация переменных в объекте (переменных экземпляра) происходит в конструкторе. В классе могут быть определены несколько конструкторов.

Синтаксис класса:

class имя\_класса{

тип\_доступа тип имя\_переменной1;

тип\_доступа тип имя\_переменной2;

…

тип\_доступа возвращаемый\_тип

имя\_метода1(список\_параметров) {тело\_метода}

}

где тип\_доступа может быть public, private, protected, internal.

Члены класса с типом доступа public доступны везде за пределами данного класса, с типом доступа protected – внутри членов данного класса и производных, с типом доступа private - только для других членов данного класса. Тип доступа internal применяется для типов, доступных в пределах одной сборки.

Пример:

class Animal{

public string Name;

private int Weight;

protected int Type;

public int Animal(int W, int T, string N){

Weight=W;

Type=T;

Name=N;

}

public int GetWeight(){return Weight;}

}

Создание обьекта

имя\_класса имя\_обьекта = new имя\_класса();

При создании обьекта класса происходит вызов соответствующего конструктора класса.

**Конструктор и деструктор**

Конструктор класса – метод для инициализации объекта при его создании. Он имеет то же имя, что и его класс. В конструкторах тип возвращаемого значения не указывается явно. Конструкторы используются для присваивания начальных значений переменным экземпляра, определенным классом, и для выполнения любых других процедур инициализации, необходимых для создания объекта.

Все классы имеют конструкторы независимо от того, определен он или нет. По умолчанию в С# предусмотрено наличие конструктора, который присваивает нулевые значения всем переменным экземпляра (для переменных обычных типов) и значения null (для переменных ссылочного типа). Но если конструктор явно определен в классе, то конструктор по умолчанию использоваться не будет.

имя\_класса(список\_параметров) {тело\_конструктора}

Деструктор – метод, вызывающийся автоматически при уничтожении обьетка класса (непосредственно перед “сборкой мусора”). Деструктор не имеет параметров и возвращаемого значения.

~имя\_класса() {тело\_деструктора}

**Наследование**

Наследование — это свойство, с помощью которого один объект может приобретать свойства другого. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз. Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса. Он может наследовать общие атрибуты от своих родительских классов.

Синтаксис:

class имя\_класса : имя\_родительского\_класса

{тело\_класса}

Пример:

class Predator:Animal{

private int Speed;

}

С помощью наследования создается иерархия классов (отношение ‘являться’). Кроме того, можно построить еще одну структуру – иерархию объектов (тогда, когда один объект является частью другого – отношение ‘часть-целое’).

**Теоретические основы**

С теоретической точки зрения:

**Класс** – это тип, описывающий устройство объектов.

**Поля** – это переменные, принадлежащие классу.

**Методы** – это функции (процедуры), принадлежащие классу.

**Объект** – это экземпляр класса, сущность в адресном пространстве компьютера.

Можно сказать, что класс является шаблоном для объекта, описывающим его структуру и поведение. Поля класса определяют структуру объекта, методы класса – поведение объекта.

С точки зрения практической реализации (в самом тексте программы) класс является типом данных, а объект – переменной этого типа.

Объявление класса состоит из двух частей: объявление заголовка класса и объявление тела класса. Заголовок класса состоит из модификатора доступа, ключевого слова class и имени самого класса. Тело класса – есть конструкция, заключенная в фигурные скобки и содержащая объявление полей и методов, принадлежащих классу.

Пример объявления класса:

public class MyClass { int a; }

Объявление объекта (создание объекта как экземпляра класса) состоит из двух частей: создание переменной-ссылки на область памяти, в которой будет располагаться объект, выделение памяти для объекта и заполнение этой памяти начальными значениями, иначе говоря инициализация данной переменной-ссылки. Объявление переменной-ссылки, а иными словами объекта, подчиняется общему правилу объявления переменных в C#. Переменные могут объявляться в любом месте в теле методов, за исключением тел условных операторов. Переменная, объявленная вне тела метода, но внутри тела класса, становится полем. Синтаксис объявления переменных имеет вид: <имя\_типа>

<имя\_переменной>.

Пример объявления целочисленной переменной:

int a;

Пример объявления переменной-объекта класса MyClass:

MyClass MyObj;

При объявлении переменная может быть сразу инициализирована (ей может быть присвоено какое-либо значение).

Пример инициализации переменной при объявлении:

int a=0;

Выделение памяти осуществляет оператор new, а задачу заполнения памяти начальными значениями решает специальный метод объекта, называемый конструктором. Конструктор – метода объекта, объявленный следующим образом: для этого метода всегда используется модификатор доступа public, нет типа возвращаемого значения (нет даже void), имя метода совпадает с именем класса. Однако компилятор С# не требует обязательного определения конструктора для класса. Если конструктор не объявлен, компилятор вызовет так называемый конструктор по умолчанию, который создаст сам.

Таким образом, создание объекта класса MyClass будет иметь вид:

MyClass MyObj=new MyClass();

Классы (и объекты) являются прямым воплощением такой идеи объектно-ориентированного программирования как инкапсуляция данных и кода.

Инкапсуляция – это механизм объединения данных и кода, манипулирующего этими данными, а также защиты того и другого от внешнего вмешательства, неправильного использования или от несанкционированного доступа. Объект – это то, что поддерживает инкапсуляцию (объединяет в себе данные и код, работающий с ними). Для закрытия данных внутри объекта используются модификаторы доступа, в частности модификатор private, который используется по умолчанию, если не употреблен другой. Данные или методы, объявленные с этим модификатором будут недоступны вне класса (объекта), то есть к ним можно будет обратиться через открытые (объявленные с модификатором public) методы или свойства класса. Свойства класса – нечто среднее между полем и методом, представляет собой конструкцию вида:

<Модификатор доступа> <Тип свойства> <Имя свойства>

{

get{return <значение>}

set{<поле>=value}

}

Обычно свойства связываются с закрытыми полями класса и помогают осуществить доступ к этим полям из внешних (относительно класса) частей программы. Свойства вместе с модификаторами доступа реализуют механизм защиты данных от несанкционированного доступа. Как мы видим, свойство имеет заголовок и тело. В заголовке указывается модификатор доступа (обычно public), тип возвращаемого свойством значения и имя свойства. В теле объявлено два метода get и set. Больше ничего в теле свойства объявлять нельзя. Метод get имеет ключевое слово return и возвращает какое-либо значение (обычно значение какого-либо поля, хотя не обязательно). Метод set имеет ключевое слово value и присваивает (устанавливает) это значение полю объекта.

Пример объявление свойства в классе MyClass:

public class MyClass

{

int a; //поле

public int A// свойство

{

get { return a;}

set { a=value;}

}

}

Пример использование описанного свойства в программе:

MyClass MyObj=new MyClass();

MyObj.A=6; // полю a объекта MyObj присвоится значение 6.

int b=MyObj.A; // переменной b присвоится значение поля a объекта MyObj.

Так как программа на языке C# может иметь множество классов со множеством методов, то необходимо каким-то образом определять точку, откуда начнется выполнятся программа. Эта точка называется точкой входа и представляет собой метод любого класса, объявленный с заголовком static void Main(string[] args). Точка входа может принадлежать любому классу, из описанных в программе, единственно, что она должна быть в программе одна.

**Задание к лабораторной работе**

В рамках консольного приложения создать класс А с полями а и b и свойством с. Свойство – значение выражения над полями а и b (выражение и типы полей – см. вариант в таблице 1). Поля инициализировать при объявлении класса. Конструктор оставить по умолчанию. Проследить, чтобы поля а и b напрямую в других классах были недоступны. Создать класс Programm с одним методом – точкой входа. В теле метода создать объект класса А, вывести на экран значение свойства с.

Конструктор – специальный метода объекта, решающий задачу начальной инициализации полей объекта и объявленный следующим образом: для этого метода всегда используется модификатор доступа public, нет типа возвращаемого значения (нет даже void), имя метода совпадает с именем класса.

Пример объявления конструктора в классе MyClass:

public class MyClass

{

int a;

public MyClass(){a=0;} //Конструктор без параметров. Инициализирует поле а нулем

public MyClass(int a){this.a=a;} //Конструктор с параметром типа int.

public MyClass(char a){this.a=a;} //Конструктор с параметром типа char.

}

Реализация для одного класса нескольких конструкторов является примером полиморфизма. Полиморфизм – механизм, позволяющий использовать одно имя для реализации схожих, но технически разных задач.

Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. В более общем смысле, в основе полиморфизма лежит идея «использовать один интерфейс для множества методов». Для компилятора полиморфные функции должны различаться принимаемыми параметрами. Это различие может быть по их количеству или по их типам.

Наследование – это процесс, посредством которого один объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов (hierarchical classification). Применение иерархии классов делает управляемыми большие потоки информации. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающи его определяли. Однако при использовании наследования можно описать объект путем определения того общего класса (или классов), к которому он относится, но со специальными чертами, делающие объект уникальным.

Синтаксис наследования следующий: при описании класса-потомка его класс-предок указывается через двоеточие.

Пример определения класса-предка Dad и класса-потомка Son:

public class Dad {}

public class Son: Dad {}

При инициализации полей объектов класса-наследника необходимо также инициализировать и поля базового класса. Инициализация полей, как было сказано выше, обычно осуществляется с использованием конструктора.

Передача управления конструктору базового класса при создании объекта – представителя производного класса осуществляется посредством конструкции ...(...):base(...){...}, которая располагается в объявлении конструктора класса-наследника между заголовком конструктора и телом. После ключевого слова base в скобках располагается список значений параметров конструктора базового класса. Очевидно, что выбор соответствующего конструктора определяется типом значений в списке (возможно, пустом) параметров.

Пример:

public class Dad

{

int a;

public Dad(int s);

}

public class Son: Dad

{

public Son(int k):base(k) {}

}

Если же у базового класса не объявлено ни одного конструктора (оставлен конструктор по умолчанию) или объявлен конструктор без параметров, тогда конструкцию base можно не использовать: при ее отсутствии управление передается конструктору без параметров.

Однако при вызове конструктора можно передавать управление не только конструктору базового класса, но и другому конструктору данного класса. Это удобно в тех случаях, когда необходимо создать множество объектов, различающихся между собой каким-либо образом, но и имеющим некую общую часть. Тогда для реализации общей части можно написать какой-то общий конструктор, а уже в других конструкторах, выполняющих более детальную настройку объекта, вызывать общий. Передача управления собственному конструктору аналогична описанной выше, только вместо ключевого слова base используется ключевое слово this.

Пример:

public class Dad

{

int a;

public Dad(int s);

}

public class Son: Dad

{

public Son(int k):base(k) {}

public Son():this(10) {}

}

В рамках данной лабораторной работы согласно варианту предложено ПОВИТОРИТЬ управляющие операторы языка C#. Управляющие операторы применяются в рамках методов. Они определяют последовательность выполнения операторов в программе и являются основным средством реализации алгоритмов.

В данной лабораторной работе необходимо использовать следующие категориями управляющих операторов:

1. Условные операторы. Вводятся ключевыми словами if, if ... else ..., switch.

2. Циклы. Вводятся ключевыми словами while, do ... while, for, foreach.

Условный оператор if имеет следующие правила использования. После ключевого слова if располагается взятое в круглые скобки условное выражение (булево выражение), следом за которым располагается оператор (блок операторов) произвольной сложности. Далее в операторе if ... else ... после ключевого слова else размещается еще один оператор.

Невозможно построить оператор if ... else ... на основе одиночного оператора объявления:

if (true) int X = 12;

if (true) int X = 12; else int Z = 1;

Такие конструкции ошибочные, так как одиночный оператор в C# – это не блок, и ставить в зависимость от условия (пусть даже всегда истинного) создание объекта нельзя.

Но в блоках операций определена своя область видимости, и создаваемые

в них объекты, никому не мешая, существуют по своим собственным правилам.

if (true) {int X = 12;}

if (true) {int X = 12;} else {int Z = 0;}

Это правило действует во всех случаях, где какой-то оператор выполняется в зависимости от условия.

Оператор switch имеет вид:

switch(<проверяемая\_переменная>)

{

case <констаната\_1\_значение\_переменной>: <оператор>; break;

….

case <констаната\_n\_значение\_переменной>: <оператор>; break;

default: < оператор>break;

}

Пример:

int val;

switch (val)

{

case 0: Console.WriteLine(0);break;

case 1: Console.WriteLine(1);break;

default: Console.WriteLine(«Число неизвестно»);break;

}

Так как case-блоки строятся на основе одиночного оператора, то объявлять переменные в них нельзя. Ключевое слово break управляет выходом из switch. В следующем примере, если значение переменной val будет равно 0,

то на экран выведется два сообщения: 0 и 1.

int val;

switch (val)

{

case 0: Console.WriteLine(0); // нет break;

case 1: Console.WriteLine(1);break;

default: Console.WriteLine(«Число неизвестно»);break;

}

Цикл while – это цикл с предусловием. Имеет следующий синтаксис:

while (УсловиеПродолжения) Оператор

Пример

int i=0;

while(i<=10) i++;

Работает по следующему правилу: сначала проверяется условие продолжения оператора и в случае, если значение условного выражения равно true, соответствующий оператор (блок операторов) выполняется.

Цикл do ... while – цикл с постусловием. Синтаксис:

do Оператор while (УсловиеПродолжения)

Пример:

int i=0;

do

i++;

while(i<=10) ;

Разница с ранее рассмотренным оператором цикла состоит в том, что здесь сначала выполняется оператор (блок операторов), а затем проверяется условие продолжения оператора.

Цикл for – пошаговый цикл. Имеет синтаксис:

for([Выражение\_Инициализации];[Условие\_Продолжения];[Выражение\_Шага])

Выражение\_Инициализации, Условие\_Продолжения, Выражение\_Шага в могут быть пустыми, но наличие пары символов ';' в заголовке цикла обязательно.

Пример:

for(int i=0;i<10;i++) Console.WriteLine(i);

Переменные, объявленные в операторе инициализации данного цикла, не могут быть использованы непосредственно после оператора до конца блока, содержащего этот оператор.

Цикл forech является универсальным перечислителем для элементов какой-либо коллекции.

**Задание 1 к лабораторной работе (общее)**

В рамках консольного приложения разработать класс В-наследник класса А (из лабораторной работы №1) с полем d и свойством с2. Свойство с2 – результат вычисления выражения над полями a, b, d. В теле свойства использовать управляющий оператор (см. вариант в таблице 2). У класса А создать конструктор, инициализирующий его поля. Для класса В определить 2 конструктора: один – наследуется от конструктора класса А, второй – собственный. В теле программы создать объекты классов А и В, продемонстрировав работу всех конструкторов. Вывести значения свойства на экран.

**Контрольные вопросы**

1) Что понимается под термином «класс»?

2) Какие элементы определяются в составе класса?

3) Каково соотношение понятий «класс» и «объект»?

4) Что понимается под термином «члены класса»?

5) Какие члены класса Вам известны?

6) Какие члены класса содержат код?

7) Какие члены класса содержат данные?

8) Перечислите пять разновидностей членов класса специфичных для

языка C#.

9) Что понимается под термином «конструктор»?

10) Сколько конструкторов может содержать класс языка C#?

11) Приведите синтаксис описания класса в общем виде.

Проиллюстрируйте его фрагментом программы на языке C#.

12) Какие модификаторы типа доступа Вам известны?

13) В чем заключаются особенности доступа членов класса с

модификатором public?

14) В чем заключаются особенности доступа членов класса с

модификатором private?

15) В чем заключаются особенности доступа членов класса с

модификатором protected?

16) В чем заключаются особенности доступа членов класса с

модификатором internal?

17) Какое ключевое слово языка C# используется при создании

объекта?

18) Приведите синтаксис создания объекта в общем виде.

Проиллюстрируйте его фрагментом программы на языке C#.

19) В чем состоит назначение конструктора?

20) Каждый ли класс языка C# имеет конструктор?

21) Какие умолчания для конструкторов приняты в языке C#?

22) Каким значением инициализируются по умолчанию значения

ссылочного типа?

23) В каком случае конструктор по умолчанию не используется?

24) Приведите синтаксис конструктора класса в общем виде.

Проиллюстрируйте его фрагментом программы на языке C#.

25) Что понимается под термином «деструктор»?

26) В чем состоит назначение деструктора?

27) Приведите синтаксис деструктора класса в общем виде.

Проиллюстрируйте его фрагментом программы на языке C#.

28) Имеет ли деструктор параметры?

29) Возвращает ли деструктор значение?

30) Что понимается под термином «наследование»?

31) Какая классификация объектов соответствует наследованию?

32) Что общего имеет дочерний класс с родительским?

33) В чем состоит различие между дочерним и родительским

классами?

34) Приведите синтаксис описания наследования классов в общем

виде. Проиллюстрируйте его фрагментом программы на языке C#.

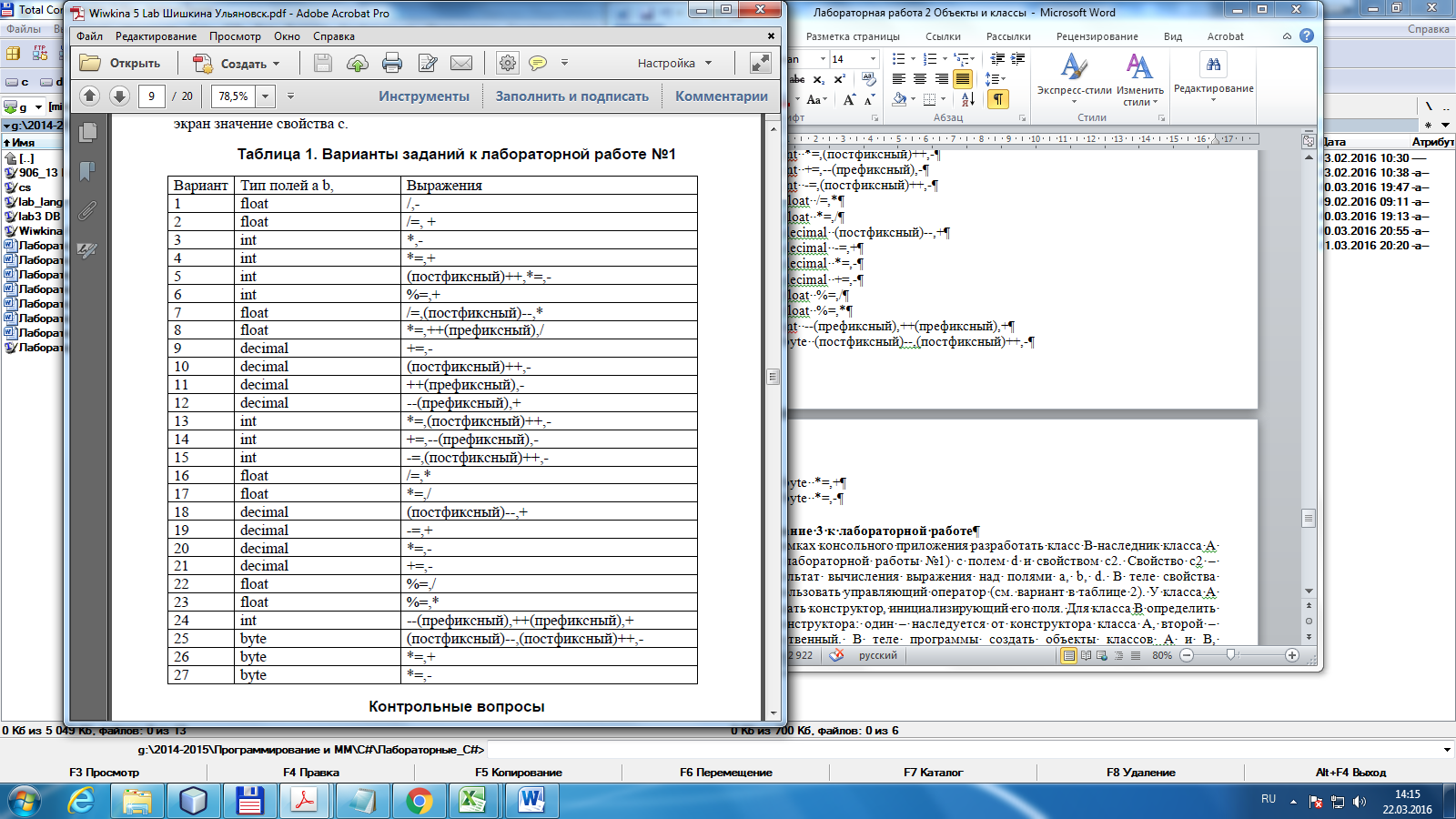
35) Какому отношению соответствует иерархия классов?

36) Какому отношению соответствует иерархия объектов?

**Задание 2 к лабораторной работе (индивидуальное)**

В рамках консольного приложения создать класс А с полями а и b и свойством с. Свойство – значение выражения над полями а и b (выражение и типы полей – см. вариант в таблице 1). Поля инициализировать при объявлении класса. Конструктор оставить по умолчанию. Проследить, чтобы поля а и b напрямую в других классах были недоступны. Создать класс Programm с одним методом – точкой входа. В теле метода создать объект класса А, вывести на экран значение свойства с.

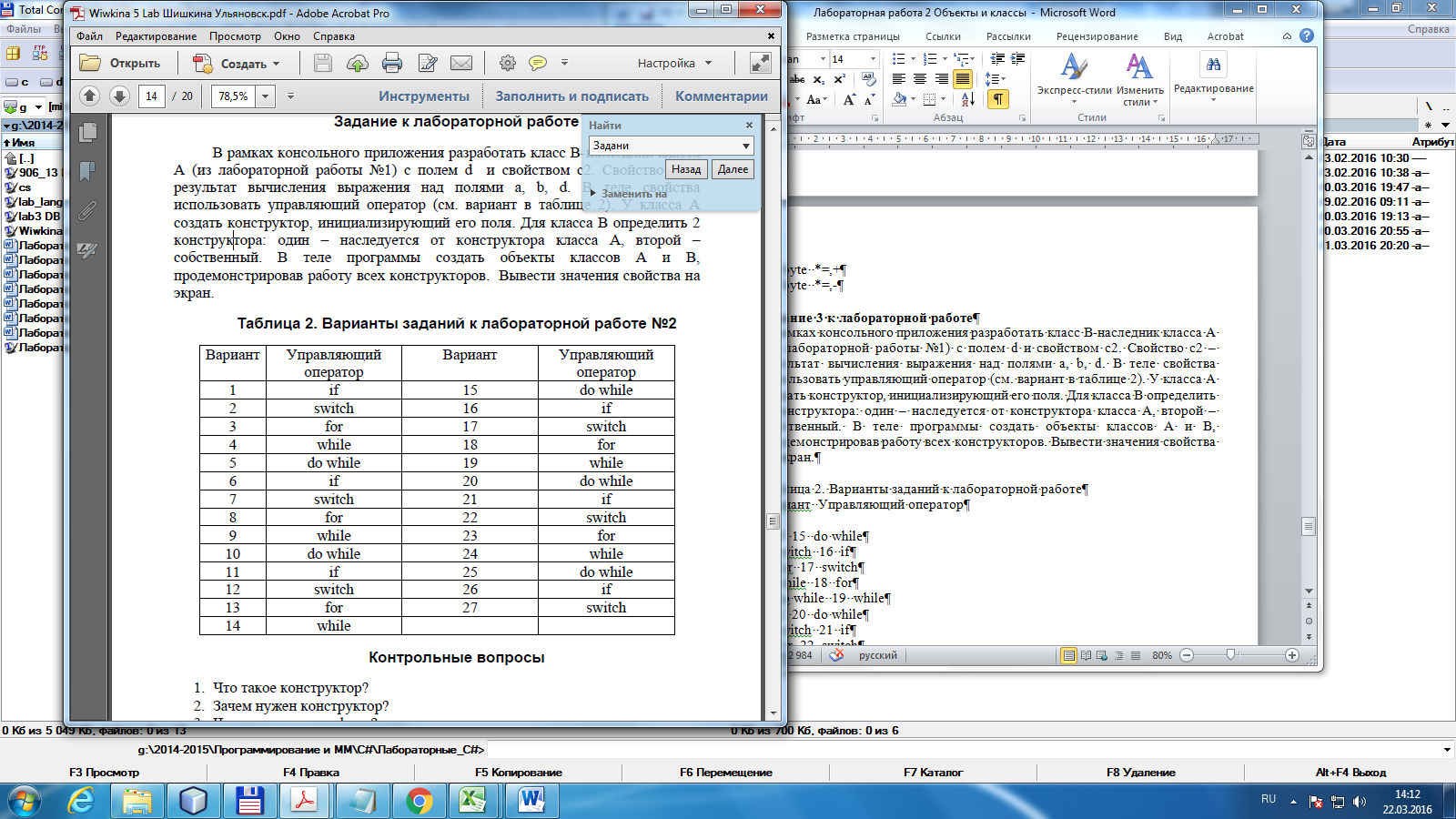
Таблица 1. Варианты заданий к лабораторной работе №1



**Задание 3 к лабораторной работе**

В рамках консольного приложения разработать класс В-наследник класса А (из лабораторной работы №1) с полем d и свойством с2. Свойство с2 – результат вычисления выражения над полями a, b, d. В теле свойства использовать управляющий оператор (см. вариант в таблице 2). У класса А создать конструктор, инициализирующий его поля. Для класса В определить 2 конструктора: один – наследуется от конструктора класса А, второй – собственный. В теле программы создать объекты классов А и В, продемонстрировав работу всех конструкторов. Вывести значения свойства на экран.

Таблица 2. Варианты заданий к лабораторной работе



**Задание 4 к лабораторной работе**

Построить иерархию классов в соответствии с вариантом задания:

1) Студент, преподаватель, персона, заведующий кафедрой

2) Служащий, персона, рабочий, инженер

3) Рабочий, кадры, инженер, администрация

4) Деталь, механизм, изделие, узел

5) Организация, страховая компания, нефтегазовая компания, завод

6) Журнал, книга, печатное издание, учебник

7) Тест, экзамен, выпускной экзамен, испытание

8) Место, область, город, мегаполис

9) Игрушка, продукт, товар, молочный продукт

10) Квитанция, накладная, документ, счет

11) Автомобиль, поезд, транспортное средство, экспресс

12) Двигатель, двигатель внутреннего сгорания, дизель, реактивный двигатель

13) Республика, монархия, королевство, государство

14) Млекопитающее, парнокопытное, птица, животное

15) Корабль, пароход, парусник, корвет

Порядок выполнения работы:

1) Разработать методы и свойства для каждого из определяемых классов.

2) Реализовать программу на C# в соответствии с вариантом исполнения.

3) Подготовить отчет в твердой копии и в электронном виде.