

**Министерство науки и высшего образования Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**  
**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**  
**ITMO University**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7**

**По дисциплине** Объектно-ориентированное программирование

**Тема работы** Создание иерархии классов

**Обучающийся** Буров Глеб Максимович

**Факультет** факультет инфокоммуникационных технологий

**Группа** K3223

**Направление подготовки** 11.03.02 Инфокоммуникационные технологии и системы связи

**Образовательная программа** Программирование в инфокоммуникационных системах

**Обучающийся**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

Буров Г.М.  
(Ф.И.О.)

**Руководитель**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

Иванов С.Е.  
(Ф.И.О.)

# СОДЕРЖАНИЕ

Стр.

<b>ВВЕДЕНИЕ .....</b>	<b>3</b>
<b>1   Ход работы .....</b>	<b>4</b>
1.1   Упражнение 1 .....	4
1.2   Упражнение 2 .....	5
1.3   Упражнение 3 .....	7
1.4   Упражнение 4 .....	9
1.5   Упражнение 5 .....	9
1.6   Упражнение 6 .....	12
1.7   Упражнение 7 .....	14
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>16</b>

## **ВВЕДЕНИЕ**

Целью данной лабораторной работы является изучение наследования как важного элемента объектно-ориентированного программирования и приобретение навыков реализации иерархии классов.

# 1 Ход работы

## 1.1 Упражнение 1

В первом упражнении нужно использовать наследование для построения иерархии между классами, имеющих отношение типа «является».

Для этого в проекте из прошлой лабораторной работы, где был класс Book, был создан новый класс Item, представляющий собой «единицу хранения» в библиотеке. Класс Item имеет два защищенных поля: invNumber (типа long, инвентарный номер) и taken (типа bool, хранит состояние объекта — взят ли на руки).

В классе Item были созданы следующие публичные методы: Take («взять» на руки), Return («вернуть» в библиотеку), GetInvNumber (возвращает инвентарный номер единицы хранения), IsAvaible (возвращает логическое значение — истина, если этот предмет имеется в библиотеке) (рис. 1.1).

```
class Item
{
    protected long invNumber;
    protected bool taken;

    Ссылка: 0
    public bool IsAvaible()
    {
        if (taken == true) return true;
        else
            return false;
    }

    Ссылка: 0
    public long GetInvNumber()
    {
        return invNumber;
    }

    Ссылка: 0
    public void Take()
    {
        taken = false;
    }

    Ссылка: 2
    public void Return()
    {
        taken = true;
    }
}
```

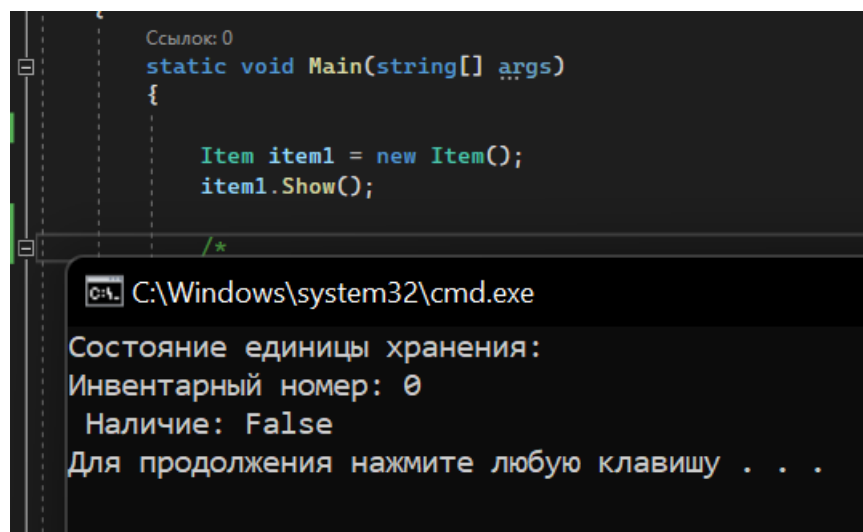
Рисунок 1.1 — Код упражнения 1

Также был реализован метод Show, отображающий информацию о единице хранения.

Далее после имени класса Book нужно было указать имя базового класса Item, чтобы реализовать отношение наследования. В определении метода Show класса Book указываем ключевое слово new, чтобы явно указать на факт скрытия метода базового класса.

Также был реализован метод TakeItem, имитировавший выдачу книг.

После этого в методе Main класса Program был создан экземпляр класса Item и у него был вызван метод Show (рис. 1.2).



```
Ссылка: 0
static void Main(string[] args)
{
    Item item1 = new Item();
    item1.Show();
}

/*
C:\Windows\system32\cmd.exe
Состояние единицы хранения:
Инвентарный номер: 0
Наличие: False
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1.2 — Работа программы из упражнения 1

## 1.2 Упражнение 2

В текущем упражнении были реализованы конструкторы у класса Item, а в производном классе Book был добавлен конструктор со ссылкой на конструктор базового класса (рис. 1.3).

```

16 private static double price = 9;
17 private bool returnSrok;
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
Ссылка: 8
abstract class Item
{
    protected long invNumber;
    protected bool taken;
    Ссылка: 3
    public Item(long invNumber, bool taken)
    {
        this.invNumber = invNumber;
        this.taken = taken;
    }
    Ссылка: 1
    public Item()
    {
        this.taken = true;
    }
}
Ссылка: 0
public Book(String author, String title, String publisher, int pages, int year,
    long invNumber, bool taken) : base(invNumber, taken)
{
    this.author = author;
    this.title = title;
    this.publisher = publisher;
    this.pages = pages;
    this.year = year;
}
Ссылка: 0
public Book()
{
}

```

Рисунок 1.3 — Конструкторы в базовом и производном классах

В метод Show класса Book был добавлен вызов метода Show класса Item (base.Show()). После этого был создан экземпляр класса Book с переданными в конструктор параметрами, у него были вызваны методы TakeItem и Show (рис. 1.4).

```

Book b2 = new Book("Толстой Л.Н.", "Война и мир", "Наука и жизнь", 1234, 2013, 101, true);
b2.TakeItem();
b2.Show();

```

```

C:\Windows\system32\cmd.exe

Книга:
Автор: Толстой Л.Н.
Название: Война и мир
Год издания: 2013
1234 стр.
Стоимость аренды: 10 р.
Состояние единицы хранения:
Инвентарный номер: 101
Наличие: False
Для продолжения нажмите любую клавишу . . .

```

Рисунок 1.4 — Пример работы программы из упражнения 2

Далее был создан класс Magazine, также являющийся производным от Item, у него были определены свои поля, конструкторы и метод Show (рис. 1.5).

```

Ссылка: 2
internal class Magazine : Item
{
    private String volume;
    private int number;
    private String title;
    private int year;

    Ссылка: 0
    public Magazine(String volume, int number, String title, int year, long invNumber, bool taken) :
        base(invNumber, taken)
    {
        this.volume = volume;
        this.number = number;
        this.title = title;
        this.year = year;
    }

    Ссылка: 0
    public Magazine()
    { }

    Ссылка: 0
    new public void Show()
    {
        Console.WriteLine("\nЖурнал: \n Том: {0} \n Номер: {1} \n Название: {2} \n Год выпуска: {3}",
            volume, number, title, year);
        base.Show();
    }
}

```

Рисунок 1.5 — Класс Magazine

### 1.3 Упражнение 3

В данном упражнении был реализован механизм полиморфизма с помощью виртуальных методов и их переопределения в произвольных классах.

Процессы возврата книг и журналов отличаются, поэтому метод Return в производных классах будет разным, и есть смысл в базовом классе объявить его виртуальным (ключевое слово virtual). В класс Book было добавлено новое поле returnSrok булевого типа и определен метод ReturnSrok, устанавливающий, что книга возвращена в срок. В произвольных классах Book и Magazine метод Return был переопределен с помощью ключевого слова override (рис. 1.6).

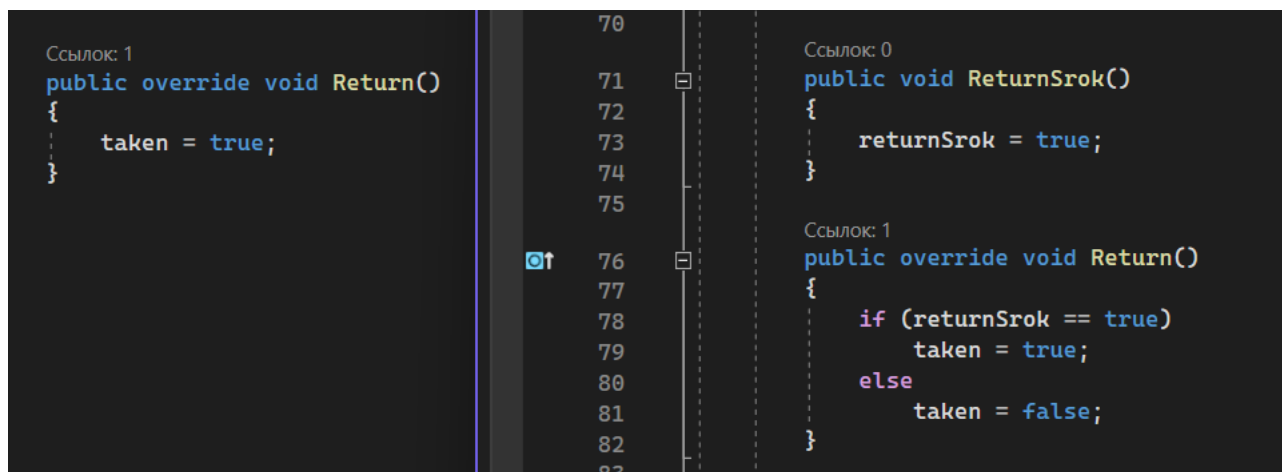


Рисунок 1.6 — Код упражнения 3

В методе Main класса Program объявлена ссылка на объект базового класса. Этой ссылке был присвоен объект производного класса — книга b2 (этот объект был ранее создан) и от имени переменной базового класса были вызваны виртуальные, что привело к вызову методов производного класса (рис. 1.7).

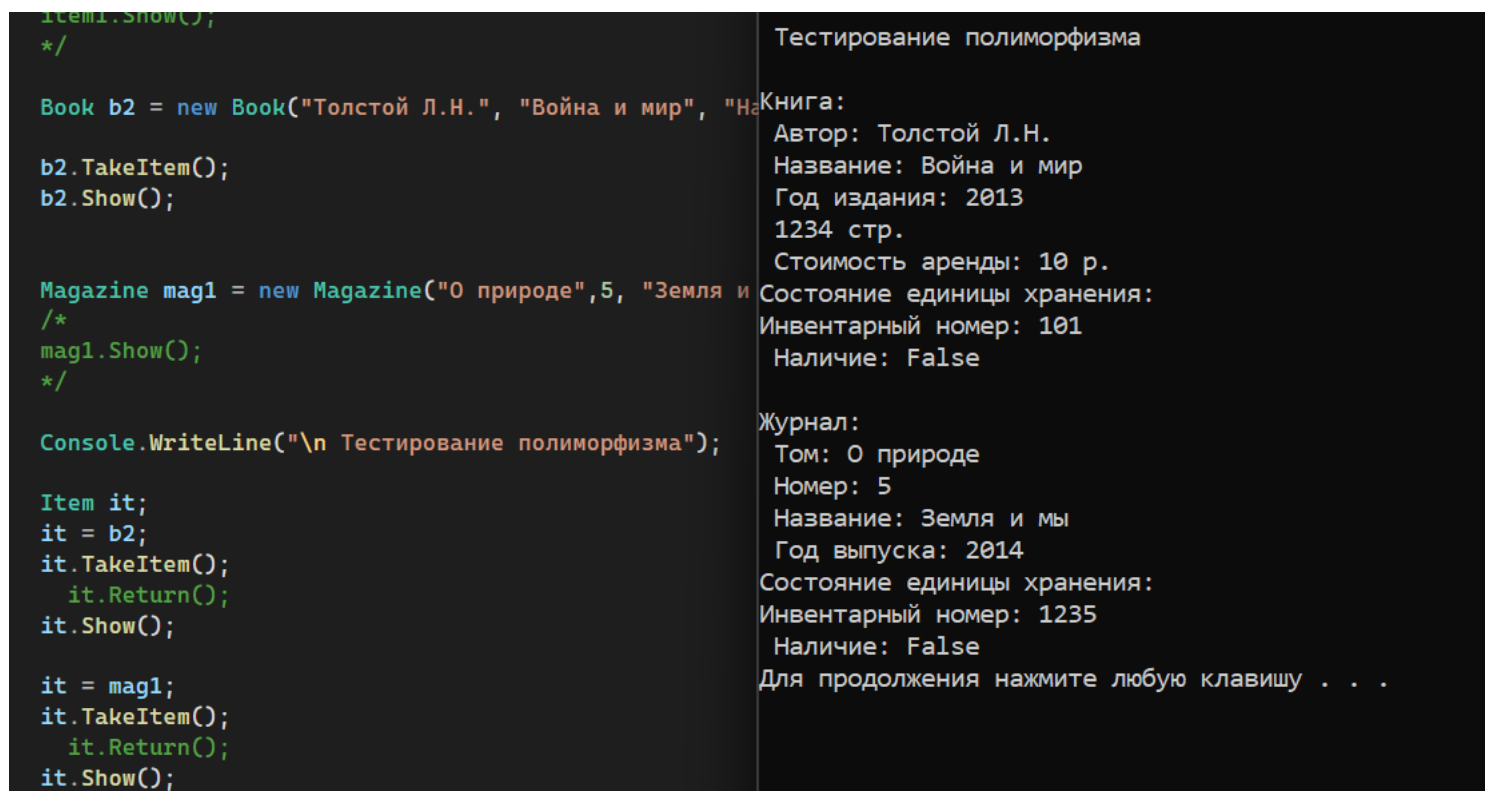


Рисунок 1.7 — Работа программы из упражнения 3



## 1.4 Упражнение 4

Поскольку создавать объекты класса `Item` не имеет смысла, его можно сделать абстрактным. Также абстрактным был объявлен метод `Return` базового класса (рис. 1.8).

```
Ссылка: 7
abstract class Item
{
    protected long invNumber;
    protected bool taken;
    Ссылка: 2
    abstract public void Return();
    Ссылка: 2
}
```

Рисунок 1.8 — Код упражнения 4

## 1.5 Упражнение 5

В этом упражнении необходимо было реализовать отношение «имеет», известное под названием модели включения или агрегации.

Для этого был создан новый проект `MyClassLine` и класс `Point`. В классе `Point` два приватных поля — координаты точки. Реализованы два конструктора (один из которых по умолчанию), метод `Show`, выводящий информацию о точке (её координаты), метод `Dlina`, рассчитывающий расстояние между текущим и переданным экземпляром класса `Point`, а также переопределён метод `ToString` для отображения объекта (рис. 1.9).

```

class Point
{
    private double x;
    private double y;

    Ссылка: 1
    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    Ссылка: 1
    public Point()
    { }

    Ссылка: 2
    public void Show()
    {
        Console.WriteLine("Точка с координатами: ({0}, {1})", x, y);
    }

    Ссылка: 1
    public double Dlina(Point p)
    {
        double dl = Math.Sqrt((this.x - p.x) * (this.x - p.x) + (this.y - p.y) * (this.y - p.y));
        return dl;
    }

    Ссылка: 0
    public override string ToString()
    {
        string ss = x + " ; " + y;
        return ss;
    }
}

```

Рисунок 1.9 — Класс Point

Далее был реализован новый класс с именем Line. Между классами Point и Line можно определить отношения агрегации — точка является частью линии и можно сказать, что отрезок «имеет» точки — начальную и конечную.

Поэтому полями в классе Line являются два объекта: точки начальная и конечная.

В классе Line были реализованы конструкторы, метод Show, выводивший на экран консоли информацию об отрезке, и метод DlinL, рассчитывающий длину отрезка с помощью метода Dlina класса Point (рис. 1.10).

```

class Line
{
    private Point pStart;
    private Point pEnd;
    Ссылка: 1
    public Line(Point pStart, Point pEnd)
    {
        this.pStart = pStart;
        this.pEnd = pEnd;
    }

    Ссылка: 0
    public Line()
    { }

    Ссылка: 1
    public void Show()
    {
        Console.WriteLine("Отрезок с координатами: ({0}) - ({1})", pStart, pEnd);
    }

    Ссылка: 1
    public double DlinL()
    {
        return pStart.Dlina(pEnd);
    }
}

```

Рисунок 1.10 — Класс Line

В методе Main класса Program была протестирована модель включения (рис. 1.11).

```

Ссылка: 0
static void Main(string[] args)
{
    Point p1 = new Point();
    p1.Show();
    Point p2 = new Point(12, 13);
    p2.Show();

    Line line = new Line(p1, p2);
    line.Show();

    double dtr = line.DlinL();
    Console.WriteLine("Длина отрезка " + dtr);
}

```

C:\Windows\system32\cmd.exe

Точка с координатами: (0, 0)  
 Точка с координатами: (12, 13)  
 Отрезок с координатами: (0 ; 0) - (12 ; 13)  
 Длина отрезка 17,6918060129541  
 Для продолжения нажмите любую клавишу . . .

Рисунок 1.11 — Работа программы из упражнения 5

## 1.6 Упражнение 6

В шестом упражнении было реализовано отношение типа ассоциация. Для этого в новом проекте Igra были созданы классы IgralnayaKost и Gamer.

Единственным полем класса IgralnayaKost является объект класса Random. Имеется также метод Random, в котором генерируется новое случайное число от 1 до 6 (рис. 1.12).

```
internal class IgralnayaKost
{
    Random r;
    Ссылка: 1
    public IgralnayaKost()
    {
        r = new Random();
    }

    Ссылка: 1
    public int random()
    {
        int res = r.Next(6) + 1;
        return res;
    }
}
```

Рисунок 1.12 — Класс IgralnayaKost

В классе Gamer два поля: имя (тип данных string) и объект класса IgralnayaKost. В классе реализованы конструктор, метод SeansGame, который возвращает значение, полученное с помощью метода Random игровой кости, и переопределен метод ToString для отображения объекта (рис. 1.13).

```

internal class Gamer
{
    string Name;
    IgralnayaKost seans;

    Ссылка: 1
    public Gamer(string name)
    {
        Name = name;
        seans = new IgralnayaKost();
    }

    public int SeansGame()
    {
        return seans.random();
    }

    — ссылки
    public override string ToString()
    {
        return Name;
    }
}

```

Рисунок 1.13 — Класс Gamer

В методе Main с помощью цикла for были имитированы несколько бросков игроком кости (рис. 1.14).

```

Ссылка: 0
static void Main(string[] args)
{
    Gamer g1 = new Gamer("Niko");
    for (int i = 1; i <= 6; i++)
    {
        Console.WriteLine("Выпало количество очков {0} для игрока {1}", g1.SeansGame(),
            g1.ToString());
    }
}

```

C:\Windows\system32\cmd.exe

```

Выпало количество очков 5 для игрока Niko
Выпало количество очков 4 для игрока Niko
Выпало количество очков 2 для игрока Niko
Выпало количество очков 2 для игрока Niko
Выпало количество очков 6 для игрока Niko
Выпало количество очков 5 для игрока Niko
Для продолжения нажмите любую клавишу . . .

```

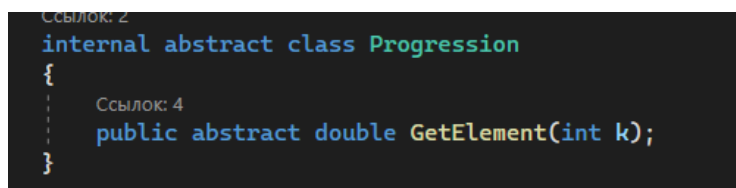
Рисунок 1.14 — Работа программы из упражнения 6

## 1.7 Упражнение 7

В последнем упражнении требовалось:

- определить абстрактный класс `Progression`, описывающий прогрессии,
- в этом классе определить абстрактный метод `GetElement` с целочисленным параметром `k`, возвращающий элемент прогрессии,
- определить два производных класса `ArithmeticProgression` и `GeometricProgression`, описывающие арифметическую и геометрическую прогрессии.
- в каждом из классов необходимо определить конструктор, задающий параметры прогрессии и перегрузить унаследованный метод `GetElement`.

Абстрактный класс `Progression` состоит только из одного абстрактного метода `GetElement` (рис. 1.15).



```
Ссылка: 2
internal abstract class Progression
{
    Ссылка: 4
    public abstract double GetElement(int k);
}
```

Рисунок 1.15 — Класс `Progression`

Классы `ArithmeticProgression` и `GeometricProgression` оба имеют поле `firstElement` (первый элемент прогрессии). Поле `diff` характерно для класса `ArithmeticProgression` (разность арифметической прогрессии), а поле `denom` — для `GeometricProgression` (знаменатель геометрической прогрессии). В обоих классах определены конструкторы (в том числе и без параметров), а также переопределен метод `GetElement` (рис. 1.16).

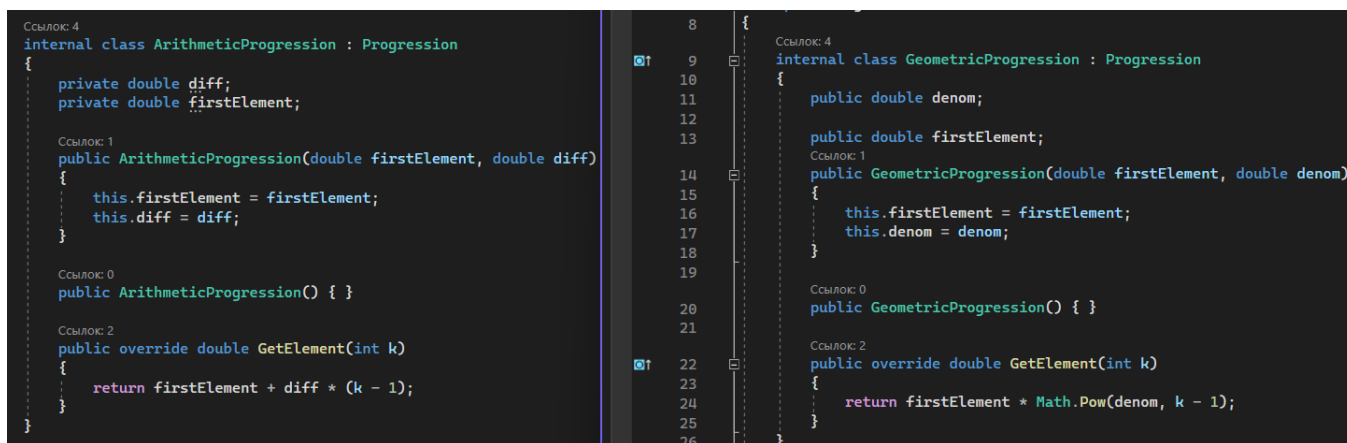


Рисунок 1.16 — Классы ArithmeticProgression и GeometricProgression

В методе Main были созданы экземпляры каждого производного класса и вызваны их методы (рис. 1.17).

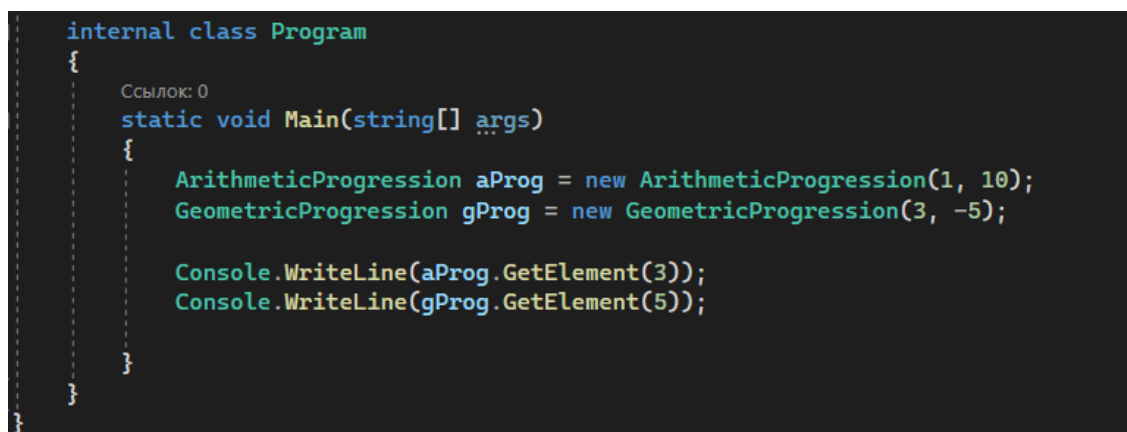


Рисунок 1.17 — Работа программы из упражнения 7

## **ЗАКЛЮЧЕНИЕ**

В ходе этой лабораторной работы я изучил различные виды отношений между классами: наследование, агрегация, ассоциация. Все они являются важными элементами объектно-ориентированного программирования. Ознакомился с тем, что такое абстрактный класс, виртуальные методы и как их переопределять. Все задания были выполнены.