Московский авиационный институт
(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы 1 — 9 по курсу ООП: основы программирования на C#

0.Перегрузка
1.Агрегация по ссылке
2.Агрегация по значению и вложением
3.Принцип подстановки
4.Наследование: расширение, спецификация, специализация, конструирование и комбинирование
5.Наследование: комбинирование через общих предков
6.Ассоциация (один к одному, один ко многим)
7.Использование
8.Конкретизация: параметров функции, метода, конструктора и атрибутов. Множественная конкретизация. Конкретизация с ограничениями (операция is).
9. Анонимные функции: сигнатура функций (delegate), Лямбда выражение(=>). Событие (Event).

Работу выполнила:
М8О-205Б-19 Данилова Татьяна Михайловна  Вариант 9

Руководитель: _____/Семенов А.С.

Дата: 28 ноября 2020 г.

Перегрузка конструкторов, функций, операторов и операций.

Текст программы:

```
using System;

namespace lab0
{
    class Reestr
    {
        public Reestr()
        {
            Console.WriteLine("This is constructor without parameters");
        }
        public Reestr(string f, int num, int old, int year)
        {
            this.f = f;
            this.num = num;
            this.old = old;
            this.year = year;
        }
        public int CurrentYear()
        {
            int CY = old + year;
            return CY;
        }
        public int CurrentYear(int N)
        {
            int CY = old + year;
            return CY + N;
        }
        public static int operator +(Reestr a, Reestr b)
        {
            return a.year + b.year;

        }
        public static int operator -(Reestr a, Reestr b)
        {
            return a.year - b.year;

        }
        public void print()
        {
            Console.Write("{0}\t", f);
            Console.Write("{0}\t", num);
            Console.Write("{0}\t", old);
            Console.Write("{0}\t", year);
            Console.WriteLine();
        }
        public void print(int num)
        {
            Console.WriteLine("Overloading operation 'print'");
            Console.WriteLine("Value of parameter: {0}", num);
        }
        string f { set; get; }
        int num, old, year;

    }
    class Program
    {
        static void Main()
        {
            Reestr a = new Reestr("Homski", 1, 1950, 55);
            Console.WriteLine("Reestr a was created");
            Reestr b = new Reestr("Ulman", 2, 1990, 15);
```

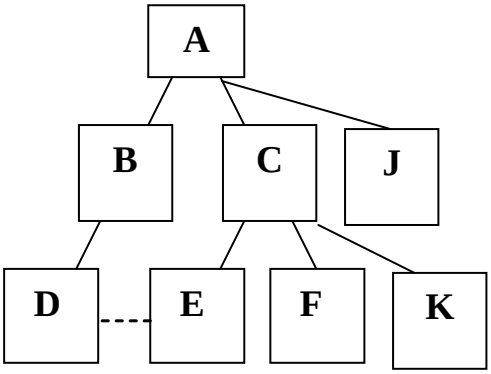
```
            Console.WriteLine("Reestr b was created");
            Reestr c = new Reestr("Lindmar", 3, 1970, 35);
            Console.WriteLine("Reestr c was created");
            Console.WriteLine();
            Console.WriteLine("Constructor with parameters and operation without parameters:");
            a.print();
            b.print();
            c.print();
            Reestr d = new Reestr();
            int D = 30;
            d.print(D);
            Console.WriteLine();
            Console.WriteLine("Function CurrentYear without parameters:");
            Console.WriteLine("Current year: {0}", a.CurrentYear());
            Console.WriteLine();
            Console.WriteLine("Overloading function CurrentYear:");
            Console.WriteLine("Current year + 30: {0}", a.CurrentYear(D));
            Console.WriteLine();
            Console.WriteLine("Operator with sum: {0}", a + b);
            Console.WriteLine("Operator overload: {0}", a - b);
            Console.ReadKey();
        }
    }
}
```

Результат работы:

```
Reestr a was created
Reestr b was created
Reestr c was created

Constructor with parameters and operation without parameters:
Homski  1       1950    55
Ulman   2       1990    15
Lindmar 3       1970    35
This is constructor without parameters
Overloading operation 'print'
Value of parameter: 30

Function CurrentYear without parameters:
Current year: 2005

Overloading function CurrentYear:
Current year + 30: 2035

Operator with sum: 70
Operator overload: 40
```

Вывод: перегрузка конструкторов, функций, операторов и операций позволяет определить структурный полиморфизм.
Иногда возникает необходимость создать один и тот же метод, но с разным набором параметров. И в зависимости от имеющихся параметров применять определенную версию метода. Это помогает сократить кол-во используемых наименований методов, что очень удобно при написании объемных программ.

Лабораторная работа №1. Агрегация по ссылке



граф 9.

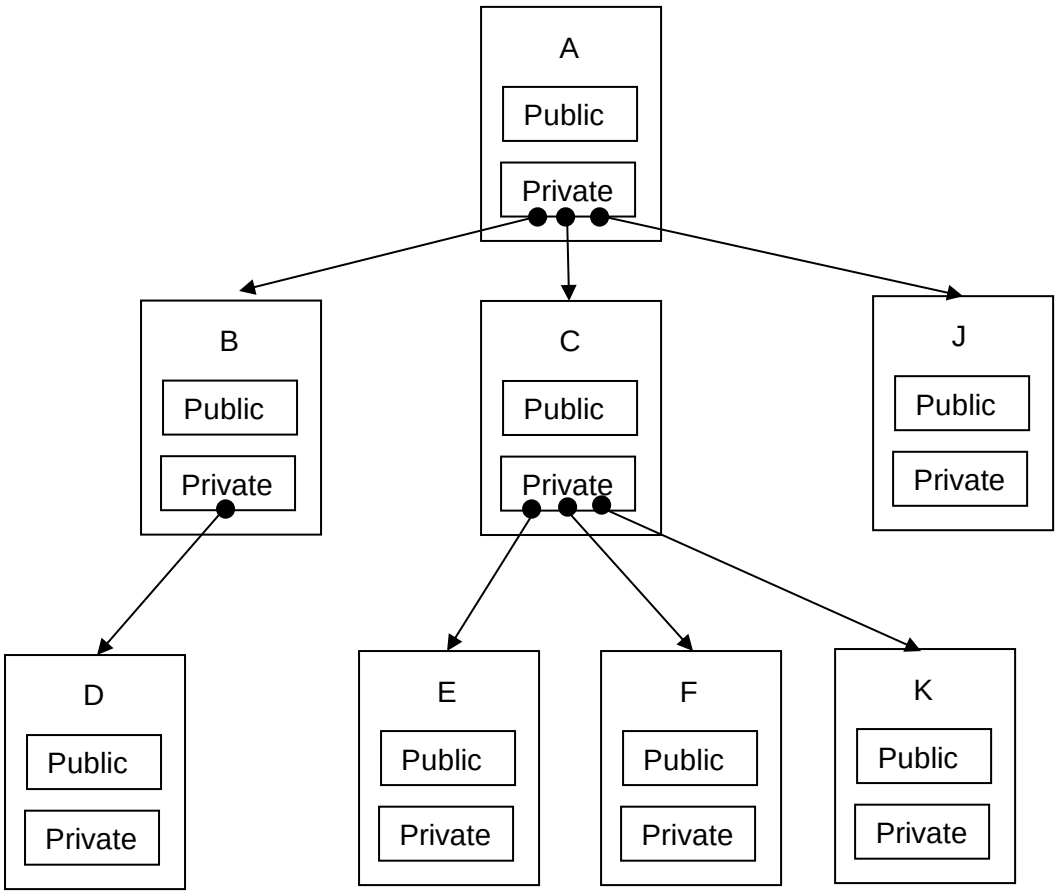
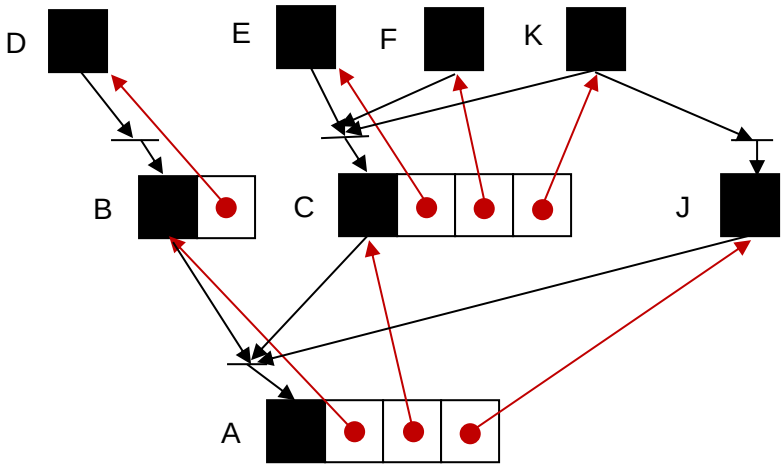


Рис.1. диаграмма классов: агрегация по ссылке.



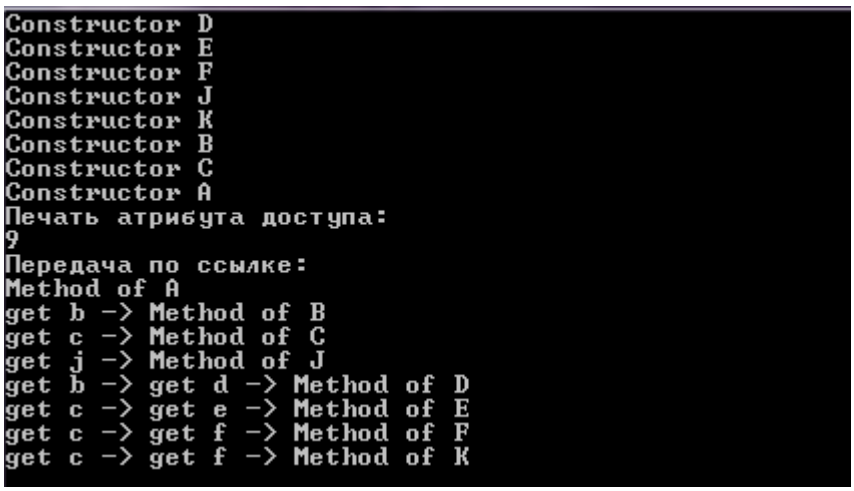
Текст программы

```
using System;

namespace lab1
{
    class A
    {
        public A(B b, C c, J j)
        {
            Console.WriteLine("Constructor A");
            this.b = b;
            this.c = c;
            this.j = j;
            c.cq = 9;
        }
        public void mA()
        {
            Console.WriteLine("Method of A");
        }
        public B bA
        {
            set { Console.WriteLine("set b"); b = value; }
            get { Console.Write("get b -> "); return b; }
        }
        public C cA
        {
            set { Console.WriteLine("set c"); c = value; }
            get { Console.Write("get c -> "); return c; }
        }
        public J jA
        {
            set { Console.WriteLine("set j"); j = value; }
            get { Console.Write("get j -> "); return j; }
        }
        private B b = null;
        private C c = null;
        private J j = null;
    }
    class B
    {
        public B(D d)
        {
            Console.WriteLine("Constructor B");
            this.d = d;
        }
```

```
    public void mB()
    {
        Console.WriteLine("Method of B");
    }
    public D dA
    {
        set { Console.WriteLine("set d"); d = value; }
        get { Console.Write("get d -> "); return d; }
    }

    private D d = null;

}
class C
{
    public C(E e, F f, K k)
    {
        Console.WriteLine("Constructor C");
        this.e = e;
        this.f = f;
        this.k = k;
    }
    public void mC()
    {
        Console.WriteLine("Method of C");
    }
    public E eA
    {
        set { Console.WriteLine("set e"); e = value; }
        get { Console.Write("get e -> "); return e; }
    }
    public F fA
    {
        set { Console.WriteLine("set f"); f = value; }
        get { Console.Write("get f -> "); return f; }
    }
    public K kA
    {
        set { Console.WriteLine("set f"); k = value; }
        get { Console.Write("get f -> "); return k; }
    }
    public int cq { get; set; }

    private E e = null;
    private F f = null;
    private K k = null;

}
class D
{
    public D() {
        Console.WriteLine("Constructor D");
    }
    public void mD()
    {
        Console.WriteLine("Method of D");
    }
}
class E
{
    public E() {
        Console.WriteLine("Constructor E");
    }
    public void mE()
    {
        Console.WriteLine("Method of E");
    }
```

```
    }
    class F
    {
        public F() {
            Console.WriteLine("Constructor F");
        }
        public void mF()
        {
            Console.WriteLine("Method of F");
        }
    }
    class J
    {
        public J() {
            Console.WriteLine("Constructor J");
        }
        public void mJ()
        {
            Console.WriteLine("Method of J");
        }
    }
    class K
    {
        public K() {
            Console.WriteLine("Constructor K");
        }
        public void mK()
        {
            Console.WriteLine("Method of K");
        }
    }
    class Program
    {
       static void Main(string[] args)
       {
            D d = new D();
            E e = new E();
            F f = new F();
            J j = new J();
            K k = new K();
            B b = new B(d);
            C c = new C(e, f, k);
            A a = new A(b, c, j);
            Console.WriteLine("Печать атрибута доступа:");
            Console.WriteLine(c.cq);
            Console.WriteLine("Передача по ссылке:");
            a.mA();
            a.bA.mB();
            a.cA.mC();
            a.jA.mJ();
            a.bA.dA.mD();
            a.cA.eA.mE();
            a.cA.fA.mF();
            a.cA.kA.mK();
            Console.ReadKey();
        }
    }
}
```

Результат работы программы:



Вывод: Объекты всех классов существуют независимо друг от друга. Связывание объектов происходит с помощью конструктора. Например, *b, c, j* — параметры для конструктора класса A. Объекты могут быть уничтожены по отдельности. Это нарушит целостность структуры. Если удалить объект *a,* объекты *b, c, j* и т.д будут продолжать существовать и дальше. Агрегация по ссылке очень выгодна, так как она расходует очень мало памяти.

Лабораторная работа №2. Агрегация по значению и вложением.

Агрегация по значению.

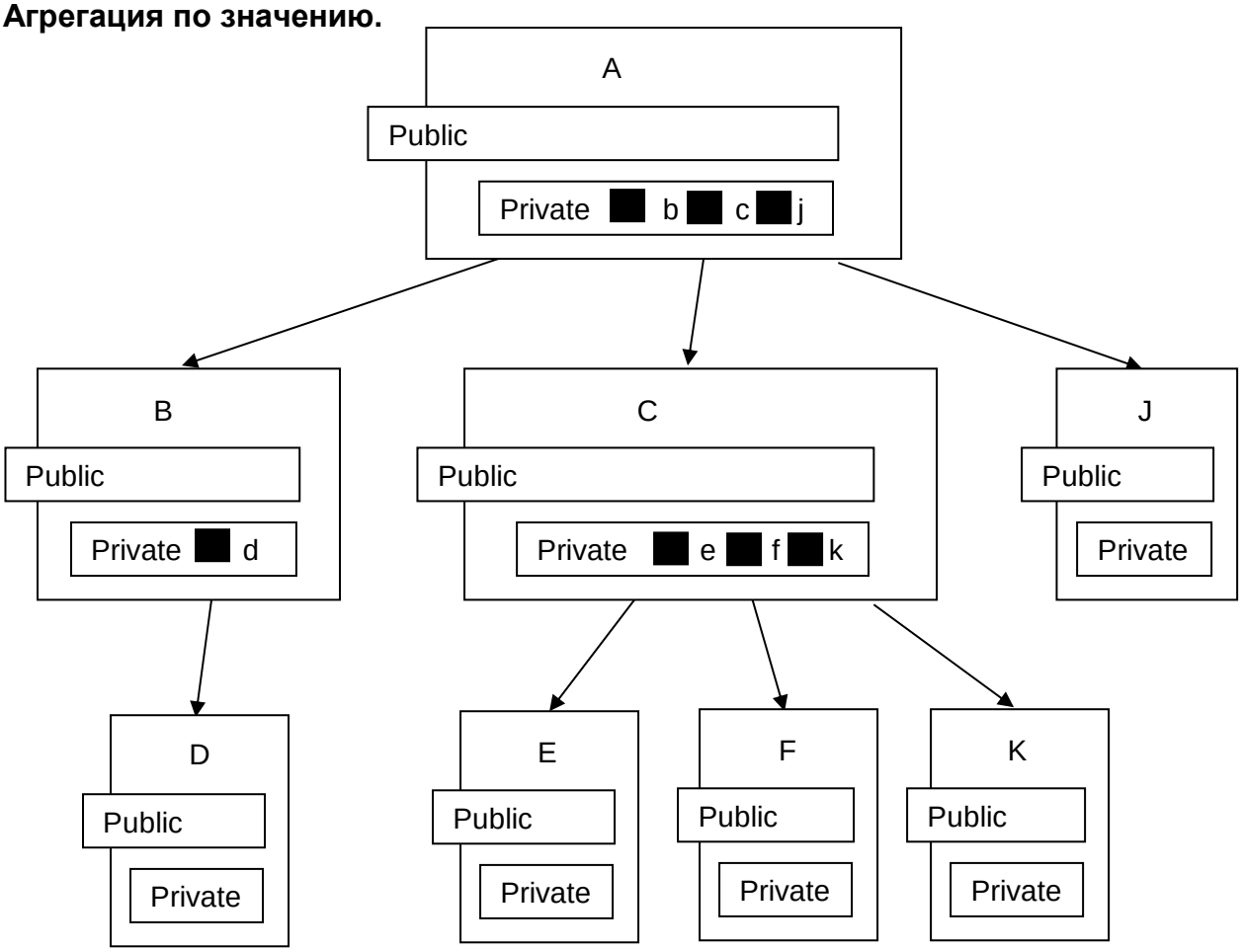
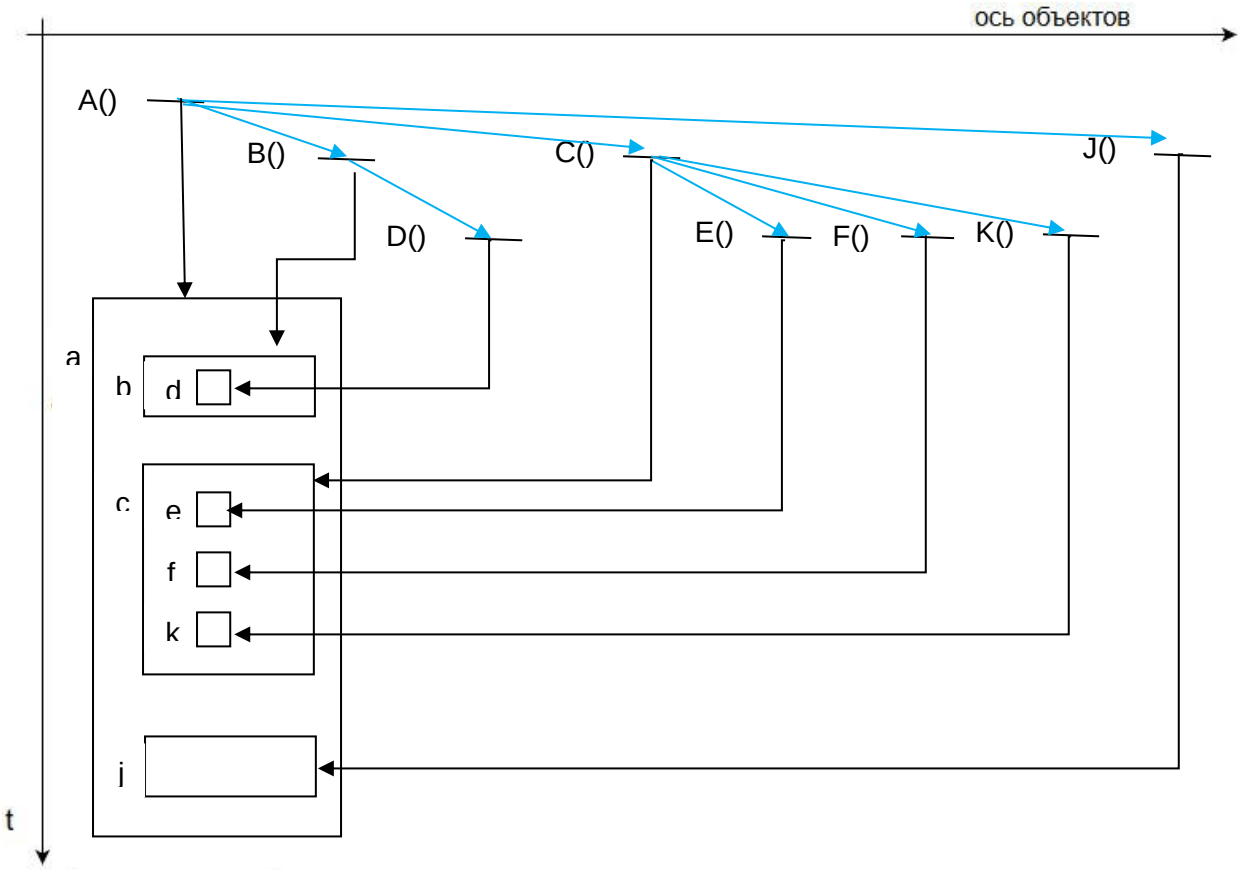


рис.2.Диаграмма классов: агрегация по значению.



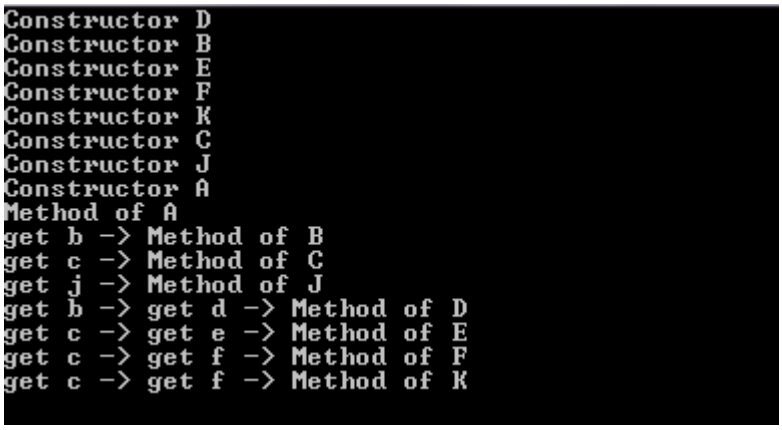
Текст программы:

```
using System;

namespace lab2._1
{
    class A
    {
        public A()
        {
            Console.WriteLine("Constructor A");
            c.c1 = 9;
        }
        public void mA()
        {
            Console.WriteLine("Method of A");
        }
        public B bA
        {
            get { Console.Write("get b -> "); return b; }
        }
        public C cA
        {
            get { Console.Write("get c -> "); return c; }
        }
        public J jA
        {
            get { Console.Write("get j -> "); return j; }
        }
        private B b = new B();
        private C c = new C();
        private J j = new J();
    }
    class B
    {
        public B() {
            Console.WriteLine("Constructor B");
        }
        public void mB()
        {
            Console.WriteLine("Method of B");
        }
        public D dA
        {
            get { Console.Write("get d -> "); return d; }
        }

        private D d = new D();

    }
    class C
    {
        public C()
        {
            Console.WriteLine("Constructor C");
            this.c1 = 15;
        }
        public void mC()
        {
            Console.WriteLine("Method of C");
        }
        public E eA
        {
            get { Console.Write("get e -> "); return e; }
        }
        public F fA
```

```
    {
        get { Console.Write("get f -> "); return f; }
    }
    public K kA
    {
        get { Console.Write("get f -> "); return k; }
    }
    public int c1 { get; set; }

    private E e = new E();
    private F f = new F();
    private K k = new K();

}
class D
{
    public D() {
        Console.WriteLine("Constructor D");
    }
    public void mD()
    {
        Console.WriteLine("Method of D");
    }
}
class E
{
    public E() {
        Console.WriteLine("Constructor E");
    }
    public void mE()
    {
        Console.WriteLine("Method of E");
    }
}
class F
{
    public F() {
        Console.WriteLine("Constructor F");
    }
    public void mF()
    {
        Console.WriteLine("Method of F");
    }
}
class J
{
    public J() {
        Console.WriteLine("Constructor J");
    }
    public void mJ()
    {
        Console.WriteLine("Method of J");
    }
}
class K
{
    public K() {
        Console.WriteLine("Constructor K");
    }
    public void mK()
    {
        Console.WriteLine("Method of K");
    }
}
class Program
{
    static void Main(string[] args)
```

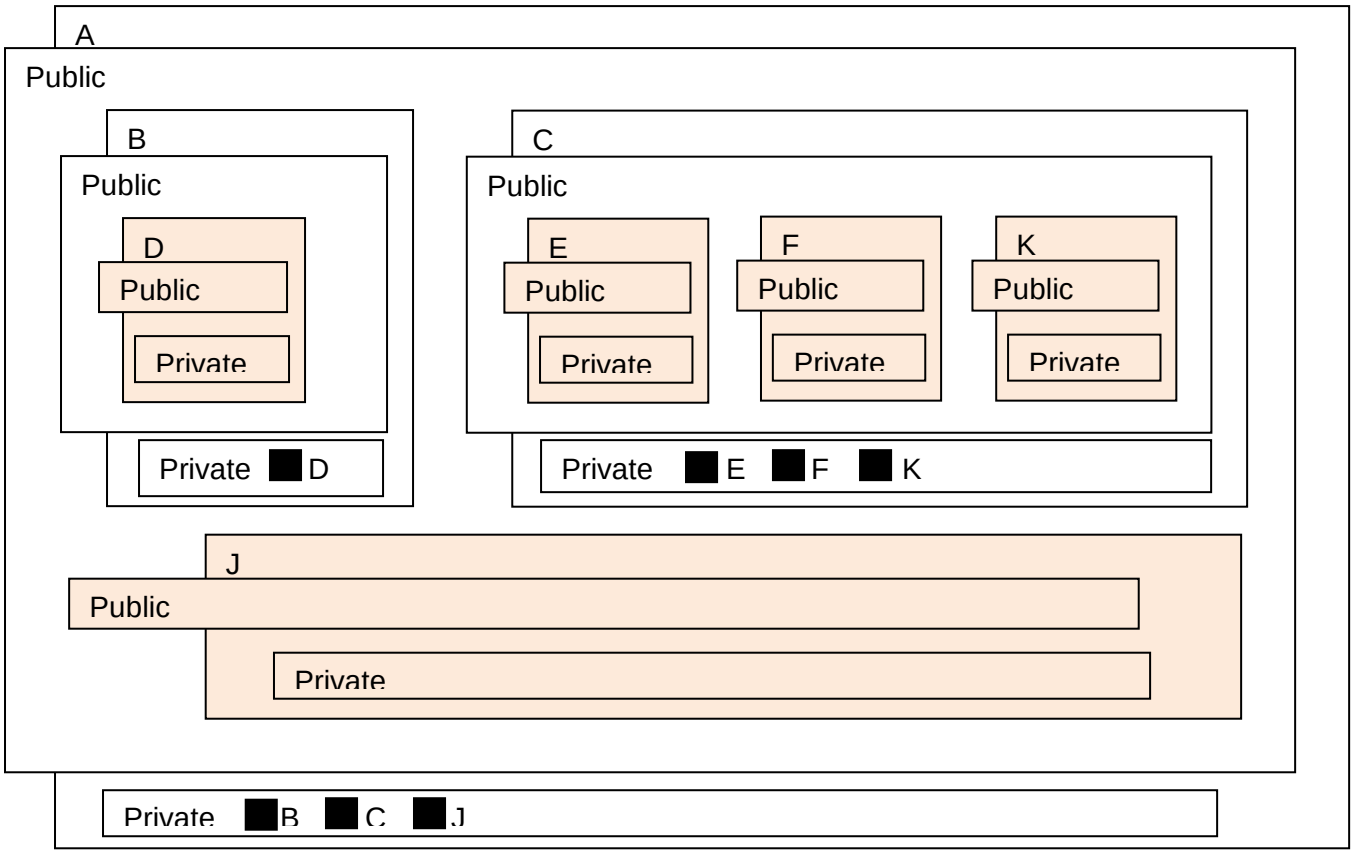
```
        {
            A a = new A();
            a.mA();
            a.bA.mB();
            a.cA.mC();
            a.jA.mJ();
            a.bA.dA.mD();
            a.cA.eA.mE();
            a.cA.fA.mF();
            a.cA.kA.mK();
            Console.ReadKey();
        }
    }
}
```

Результат работы программы:



Вывод: при агрегации по значению все объекты класса существуют внутри объявленного класса. При таком виде агрегации невозможно удалить объекты, являющиеся частью объекта первого по иерархии класса. Например, *b, c, j* — части объекта ***a*** класса A(первый класс по иерархии); эти части создаются только при вызове конструктора класса A, а уничтожаются — при вызове деструктора A.

Агрегация вложением.



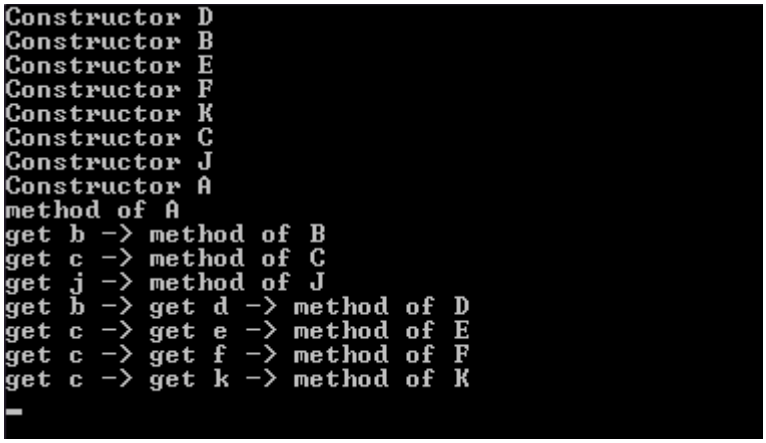
Текст программы:

```
using System;

namespace lab2._2
{
    class A
    {
        public A()
        {
            Console.WriteLine("Constructor A");
            c.c1 = 9;
        }
        public class B
        {
            public B() {
                Console.WriteLine("Constructor B");
            }
            public class D
            {
                public D() {
                    Console.WriteLine("Constructor D");
                }
                public void mD()
                {
                    Console.WriteLine("method of D");
                }
            }
            public void mB()
            {
                Console.WriteLine("method of B");
            }
```

```
    public D dA
    {
        get { Console.Write("get d -> "); return d; }
    }
    private D d = new D();
}
public class C
{
    public C()
    {
        Console.WriteLine("Constructor C");
        this.c1 = 10;
    }
    public class E
    {
        public E() {
            Console.WriteLine("Constructor E");
        }
        public void mE()
        {
            Console.WriteLine("method of E");
        }
    }
    public class F
    {
        public F() {
            Console.WriteLine("Constructor F");
        }
        public void mF()
        {
            Console.WriteLine("method of F");
        }
    }
    public class K
    {
        public K() {
            Console.WriteLine("Constructor K");
        }
        public void mK()
        {
            Console.WriteLine("method of K");
        }
    }
    public void mC()
    {
        Console.WriteLine("method of C");
    }
    public E eA
    {
        get { Console.Write("get e -> "); return e; }
    }
    public F fA
    {
        get { Console.Write("get f -> "); return f; }
    }
    public K kA
    {
        get { Console.Write("get k -> "); return k; }
    }
    private E e = new E();
    private F f = new F();
    private K k = new K();
    public int c1 { set; get; }
}
public class J
{
    public J() {
```

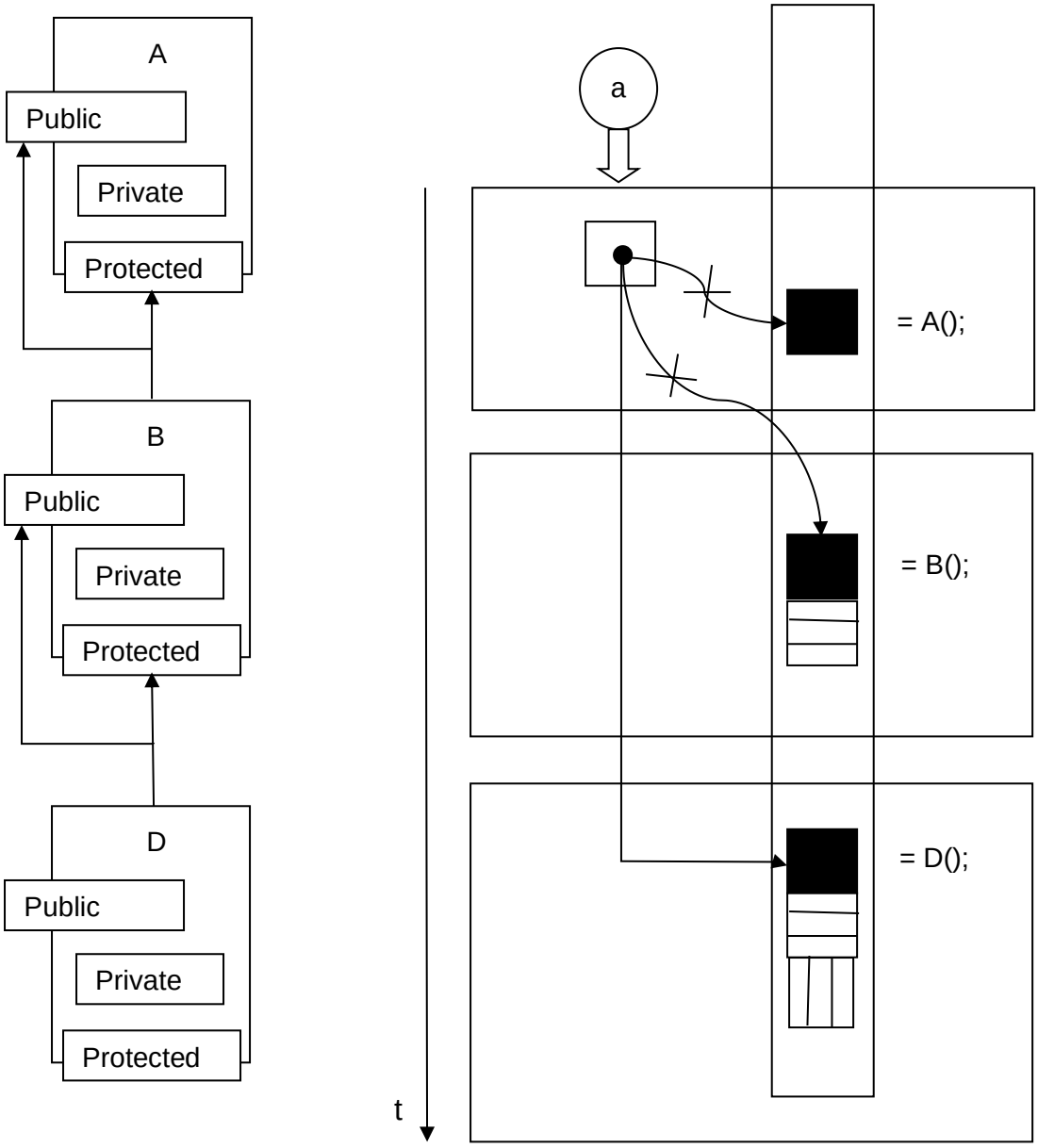
```
                Console.WriteLine("Constructor J");
            }
            public void mJ() { Console.WriteLine("method of J"); }
        }
        public void mA()
        {
            Console.WriteLine("method of A");
        }
        public B bA
        {
            get { Console.Write("get b -> "); return b; }
        }
        public C cA
        {
            get { Console.Write("get c -> "); return c; }
        }
        public J jA
        {
            get { Console.Write("get j -> "); return j; }
        }
        private B b = new B();
        private C c = new C();
        private J j = new J();
    }
    class Program
    {
        static void Main(string[] args)
        {
            A a = new A();
            a.mA();
            a.bA.mB();
            a.cA.mC();
            a.jA.mJ();
            a.bA.dA.mD();
            a.cA.eA.mE();
            a.cA.fA.mF();
            a.cA.kA.mK();
            Console.ReadKey();
        }
    }

}
```

Результат работы программы:



Вывод: при агрегации вложением определение классов происходит внутри классов, стоящих выше по иерархии. Все объекты создаваемого класса существуют внутри него самого. Как и в случае агрегации по значению уничтожение объектов, невозможно без уничтожения класса, стоящего выше по иерархии. Агрегация по ссылке намного лучше агрегации по значению или вложению, так как агрегация по ссылке расходует меньше памяти.

Лабораторная работа №3. Принцип подстановки.



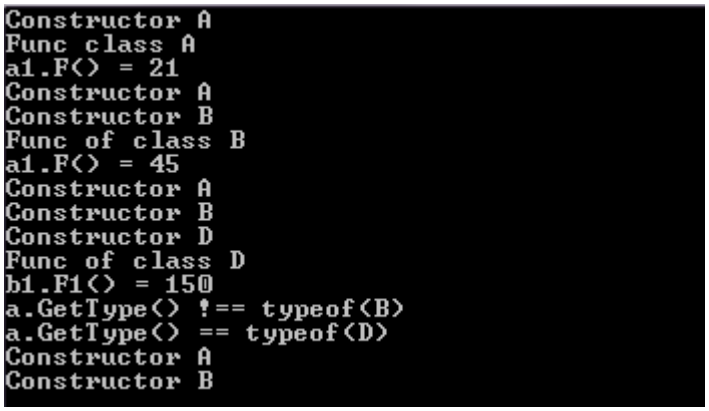
Текст программы:

```
using System;
using System.Threading;

namespace lab3
{
    class A
    {
        public A()
        {
            Console.WriteLine("Constructor A");
            this.varA = 1;
        }
        ~A() {
            Console.WriteLine("Dist A");
            Thread.Sleep(2000);
        }
        public virtual int F()
        {
```

```
        Console.WriteLine("Func class A");
        return this.varA + 20;
    }
    protected int varA { get; set; } //атрибут доступа
}

class B : A
{
    public B()
    {
        Console.WriteLine("Constructor B");
        this.varA = 5;
    }
    ~B() {
        Console.WriteLine("Dist B");
        Thread.Sleep(2000);
    }
    public override int F() //замещение
    {
        Console.WriteLine("Func of class B");
        return this.varA + 40;
    }
}

class D : B
{
    public D()
    {
        Console.WriteLine("Constructor D");
        this.varA = 50;
    }
    ~D() {
        Console.WriteLine("Dist D");
        Thread.Sleep(2000);
    }
    public override int F() //замещение
    {
        Console.WriteLine("Func of class D");
        return (this.varA + 100);
    }
}
class Program
{
    static void Main(string[] args)
    {
        A a = new A(); //a - полиморфная переменная
        Console.WriteLine("a.F() = {0}", a.F());
        Console.ReadKey();

        a = new B(); //подстановка
        Console.WriteLine("a.F() = {0}", a.F()); //замещение функции F()
        Console.ReadKey();

        a = new D(); //подстановка
        Console.WriteLine("a.F() = {0}", a.F()); //замещение функции F()
        Console.ReadKey();

        if (a.GetType() == typeof(B))
        {
            Console.WriteLine("a.GetType() == typeof(B)");
        }
        else
        {
            Console.WriteLine("a.GetType() !== typeof(B)");
            if (a.GetType() == typeof(D))
            {
                Console.WriteLine("a.GetType() == typeof(D)");
```

```
                }
                else
                {
                    Console.WriteLine("a.GetType() !== typeof(D)");
                }

            }

            {
                B b = new B(); //локальная переменная
            }
            Console.ReadKey();
            Thread.Sleep(2000);
        }
    }
}
```

Результат работы программы:



Вывод: в этой программе используется метод подстановки и метод замещения.
Принцип подстановки: вместо объекта суперкласса можно подставить объект подкласса.
Принцип замещения: функцию суперкласса можно заменить функцией подкласса.
Ключевое слово virtual используется для изменения объявлений методов, свойств, индексаторов и событий, и разрешения их переопределения в производном классе. Например, этот метод может быть переопределен любым наследующим его классом: модификатор override требуется для расширения или изменения абстрактной, или виртуальной реализации унаследованного метода, свойства, индексатора или события.

Полиморфной называется такая переменная, которая может хранить в себе значения различных типов данных.

Лабораторная работа №4. Наследование: расширение, спецификация, специализация, конструирование.

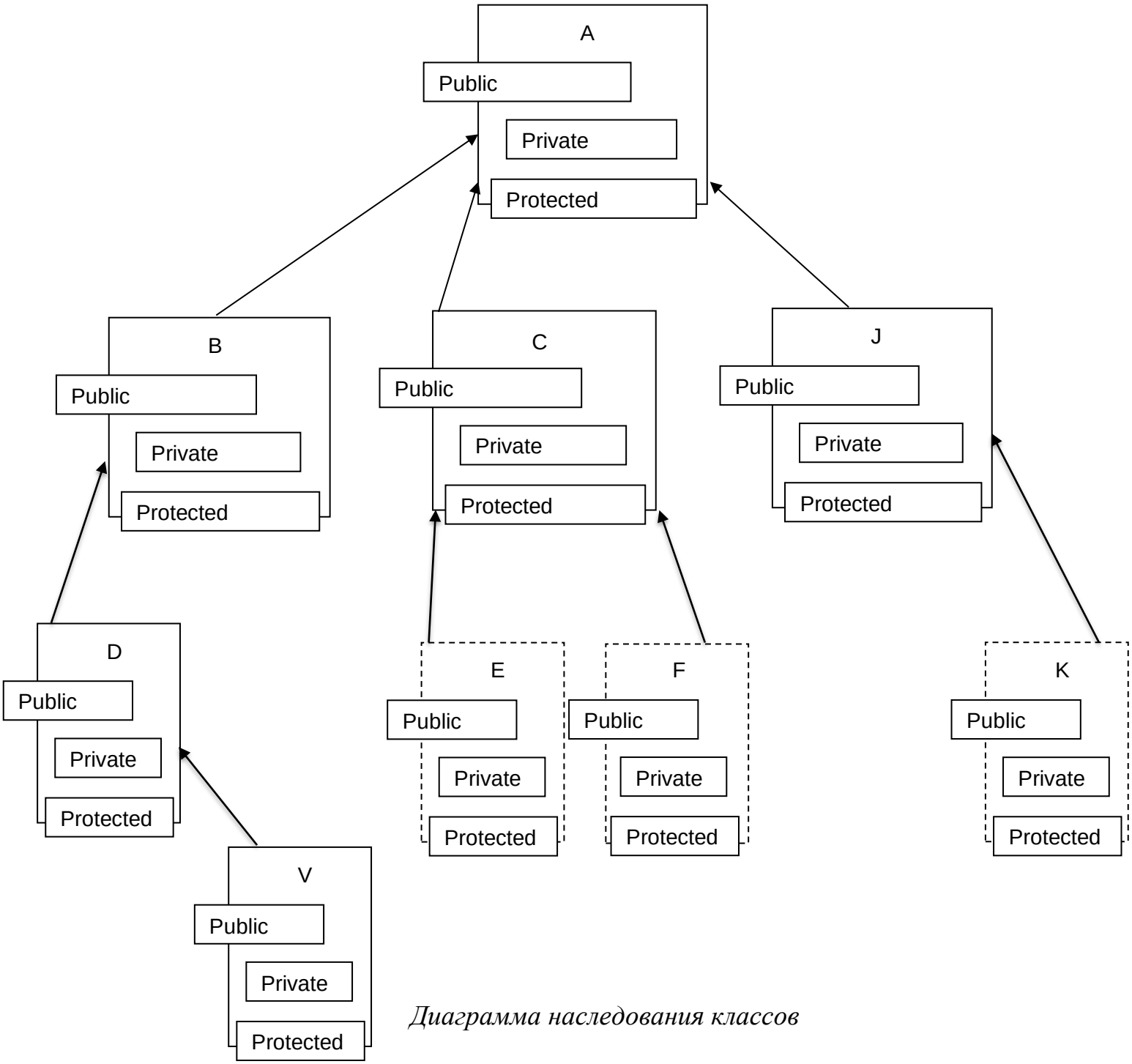


Диаграмма наследования классов

Расширение: B → A;
Спецификация: E → C, F → C(через абстрактный класс);
K → J (через интерфейс);
Специализация: D →B;
Конструирование: V→ B.

Наследование − это такое отношение между классами, когда один класс повторяет структуру и поведение другого класса (одиночное наследование) или других (множественное наследование) классов.
С помощью ключевого слова **base** мы можем обратиться к базовому классу.
При вызове конструктора класса сначала отрабатывают конструкторы базовых классов и только затем конструкторы производных.

Форма наследования **Специализация**:

- Дочерний класс является более конкретным, частным или специализированным случаем родительского класса
- Дочерний класс удовлетворяет спецификациям родителя во всех существенных моментах, т.е. его можно использовать вместо родительского класса
- Поведение базового класса, в основном, переопределяется

Форма наследования **Спецификация**

- Родительский класс описывает поведение, которое реализуется в дочернем классе, но оставлено нереализованным в родительском
- В таких случаях родительский класс называют абстрактно-специфицированным классом

Форма наследования **Расширение**:

- Дочерний класс добавляет новые функциональные возможности к родительскому классу, но не меняет наследуемое поведение
- В отличие от обобщения или специализации при расширении дочерний класс не переопределяет ни одного метода базового класса, а добавленные методы слабо связаны с существующими методами родителя

Форма наследования **Конструирование**

- Дочерний класс использует методы, предопределяемые родительским классом
- Между дочерним и родительским классами отсутствует отношение «is-a» или «быть экземпляром», то-есть дочерний класс не является более специализированной формой родительского класса
- Обычно для реализации такой формы наследования используется механизм закрытого наследования
- Дочерний класс часто изменяет не только имена методов базового класса, но и аргументы

Текст программы (4.1):

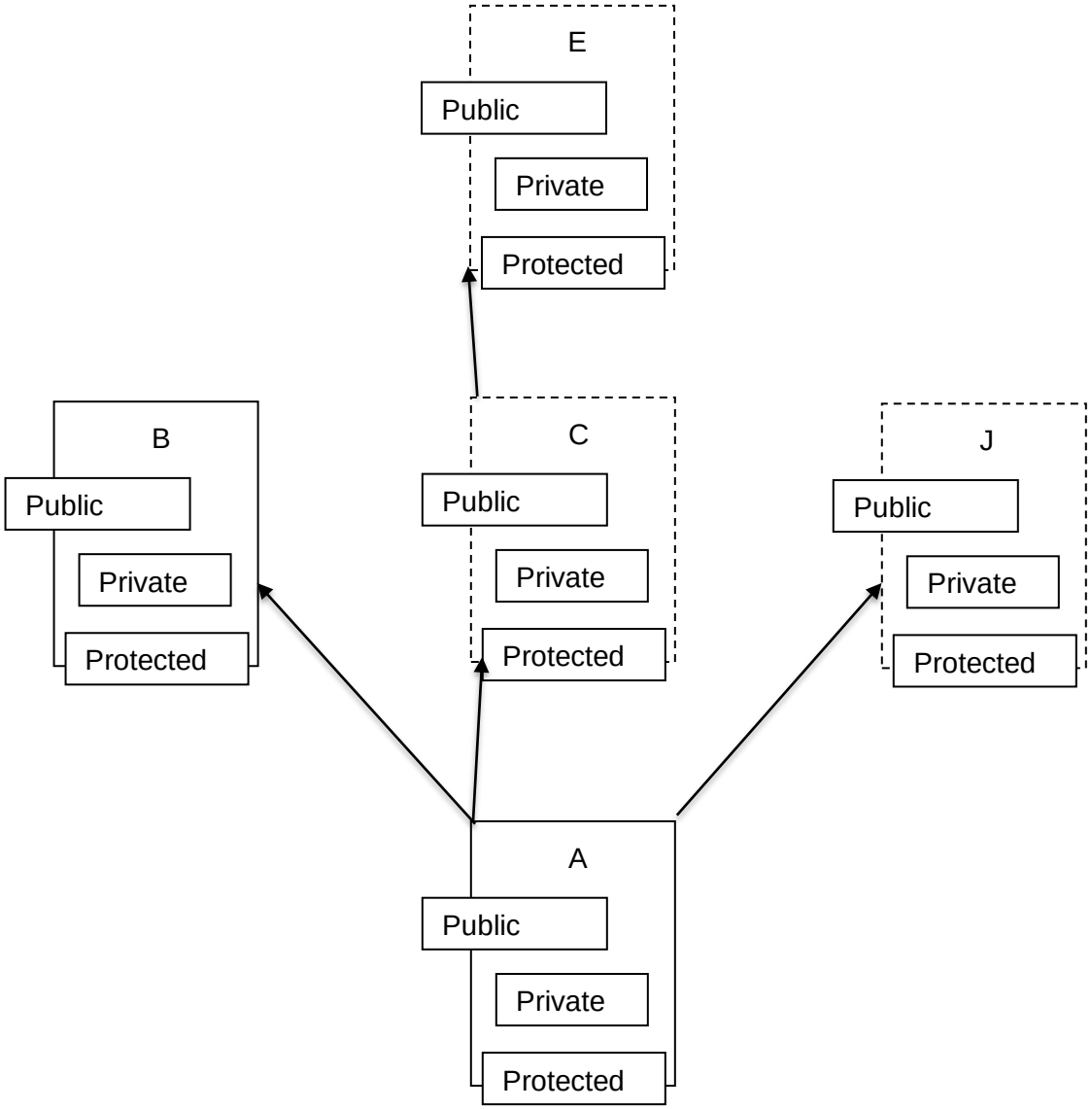
```
using System;
namespace lab_4
{
    class A //супер-класс
    {
        public A()
        {
            Console.WriteLine("Constructor A");
            this.a = 1;
        }
        ~A()
        {
            Console.WriteLine("Distructor ~A()");
            //Thread.Sleep(2000);
        }
        public virtual int fa()//virtual для замещения функции
        {
            Console.WriteLine("class A fa() ");
            return a + 1;
        }
        public int Aa
        {
            set { a = value; }
            get { Console.WriteLine("  get  "); return a; }
        }
        protected int a { set; get; } //атрибут везде наследуется (атрибут доступа по умолчанию)
    }

    class B : A
    {
        public B()
        {
            Console.WriteLine("Constructor B");
            this.a = 20;
            this.b = 10;
            this.b1 = -1;
        }
        ~B()
        {
            Console.WriteLine("distructor B");
            //Thread.Sleep(2000);
        }

        //замещение + выполнение предыдущего кода в суперклассе
        //расширение функции
        public override int fa()
        {
            Console.WriteLine("class B fa()");
            base.fa();
            return a + 10;
        }
        public int fb()
        {
            Console.WriteLine("class B fb()");
            return a + b + 10;
        }
        protected int b { set; get; }
        public int b1 { set; get; }
    }


    abstract class C
    {
        abstract public int fc_1();
        abstract public int fc_2();
```

```
    public void print() { Console.WriteLine("class C print"); }//будет просто наследоваться
    public int Cc
    {
        set { c = value; }
        get { Console.WriteLine("   get  "); return c; }
    }
    protected int c = 1;
}


class E : C
{
    public E() { this.c = 22; }
    public override int fc_1() //замещаем эту функцию из класса C
    {
        Console.WriteLine("class E fc1()");
        return c * 5;
    }

    public override int fc_2() //замещаем эту функцию из класса C
    {
        Console.WriteLine("class E fc2()");
        return c / 2;
    }
}

class F : C
{
    public F() { this.c = 1022; }
    public override int fc_1() //замещаем эту функцию из класса C
    {
        Console.WriteLine("class F fc1()");
        return c * 15;
    }

    public override int fc_2() //замещаем эту функцию из класса C
    {
        Console.WriteLine("class F fc2()");
        return c / 2 + 1000;
    }
}

interface J
{
    int fj_1();
    int fj_2();
    int fj() { return 0; }
}

class K : J
{
    public K() { }
    public int fj_1() { return 10; }
    public int fj_2() { return 20; }
    public int a { get; set; }

}

class D : B
{
    public D()
    {
        Console.WriteLine("Constructor D");
        this.a = 200;
    }
    public override int fa()
    {
```

```
            Console.WriteLine("class D fa()");
            return a + 4;
        }
    }

    class V : D
    {
        public V() {
            Console.WriteLine("Constructor D");
            base.fa();
        }
        public override int fa()
        {
            Console.WriteLine("class V fa()");
            return 0;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            A a = new A();
            //создаём обьект
            Console.WriteLine("a.fa() = {0}", a.fa());

            a = new B();
            Console.WriteLine("a.fa() = {0}", a.fa()); //Замещённая функция (вместо супер-класса
становится функцией подкласса)
            Console.WriteLine();

            Console.WriteLine("Expansion");
            Console.WriteLine("(B)a.fb() = {0}", ((B)a).fb()); //расширение по функциям
            Console.WriteLine("(B)a.b1 = {0}", ((B)a).b1); //расширение по атрибутам

            Console.WriteLine("Specification (abstract class)");
            C c = null; //создаём ссылку на обьект  и подставляем под него обьект который мы
пронаследовали
            c = new E();
            c.Cc = 455; //поработали с обьектом суперкласса
            c.print(); //пронаследовалась
            Console.WriteLine("c.fc_1() = {0}", c.fc_1()); //замещение fc_1() класса C функцией
fc() класса E
            Console.WriteLine("c.fc_2() = {0}", c.fc_2());

            c = new F();        //подставляем обьект F
            c.print();      //пронаследовалась
            Console.WriteLine("c.fc_1() = {0}", c.fc_1()); //замещение fc_1() класса C функцией
fc_1() класса F
            Console.WriteLine("c.fc_2() = {0}", c.fc_2());

            Console.WriteLine("Specification (interface)");
            J j = null;
            j = new K();
            Console.WriteLine(" j.fj_1() = {0}", j.fj_1());
            Console.WriteLine(" j.fj_2() = {0}", j.fj_2());

            Console.WriteLine("Specialization");
            a = new D();
            Console.WriteLine("a.fa() = {0}", a.fa());


            Console.WriteLine("Construction");
            a = new V();
            Console.WriteLine("a.fa() = {0}", a.fa());

            Console.ReadKey();
```

```
        }
    }
}
```

```
(B)a.b1 = -1
Specification (abstract class)
class C print
class E fc1()
c.fc_1() = 2275
class E fc2()
c.fc_2() = 227
class C print
class F fc1()
c.fc_1() = 15330
class F fc2()
c.fc_2() = 1511
Specification (interface)
 j.fj_1() = 10
 j.fj_2() = 20
Specialization
Constructor A
Constructor B
Constructor D
class D fa()
a.fa() = 204
Construction
Constructor A
Constructor B
Constructor D
Constructor V
class D fa()
class V fa()
a.fa() = 0
```



Форма наследования **Комбинирование**

- Дочерний класс наследует черты более чем одного родительского класса
- Для комбинирования классов используется механизм множественного наследования

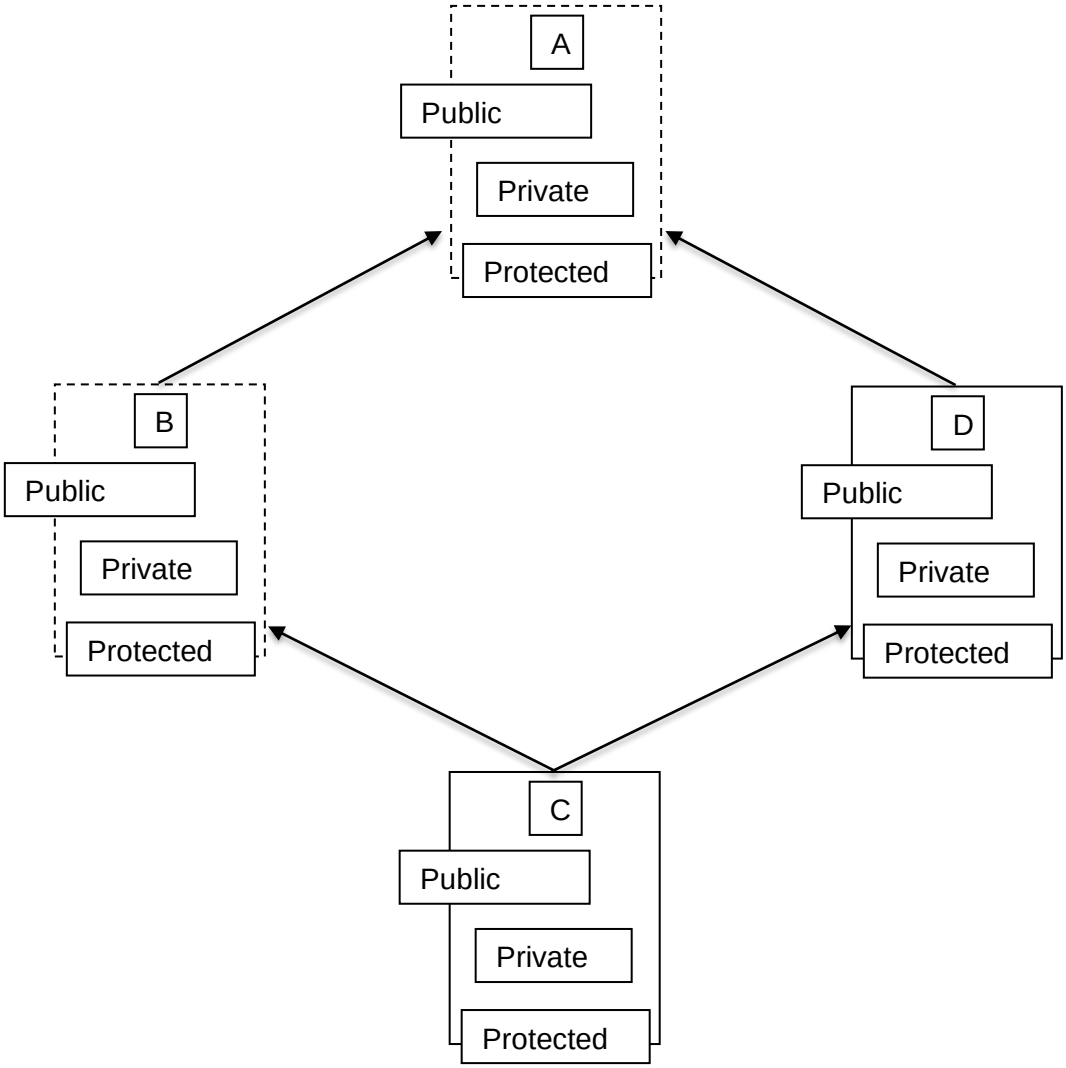
Текст программы (4.2):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab_4_2
{
    class B {
        public B() { Console.WriteLine("Constructor B"); }
        public virtual void fb() {
            Console.WriteLine("fb B");
        }
        public int b { set; get; }
    }

    interface E  {
        int fe();
```

```
    }

    interface C: E {
       int fc();
    }

    interface J  {
        int fj();
    }

    class A : B, C, J {
        public A() { Console.WriteLine("Constructor A"); }
        public override void fb() {
            base.fb();
            Console.WriteLine("fb override A");
        }
        public int fc() {   return 1; }
        public int fj() {   return 2; }
        public int fe() { return 3; }

        public int a { set; get; }
    }

    class Program
    {
        static void Main(string[] args) {

            B b = new B();
            b.b = 50;
            b = new A();
            ((A)b).fb();
            Console.ReadKey();
            Console.WriteLine();

            C c = null;
            c = new A();
            ((A)c).fb();
            Console.WriteLine();
            A a = new A();
            a.fb();
            Console.WriteLine();
            J j = null;
            j = new A();
            Console.WriteLine(" j.fj() {0}", j.fj());
            Console.ReadKey();
            Console.WriteLine(" c.fe() {0}", c.fe());
            Console.ReadKey();
        }
    }
}
```

```
Constructor B
Constructor B
Constructor A
fb B
fb override A

Constructor B
Constructor A
fb B
fb override A

Constructor B
Constructor A
fb B
fb override A

Constructor B
Constructor A
 j.fj() 2
 c.fe() 3
```

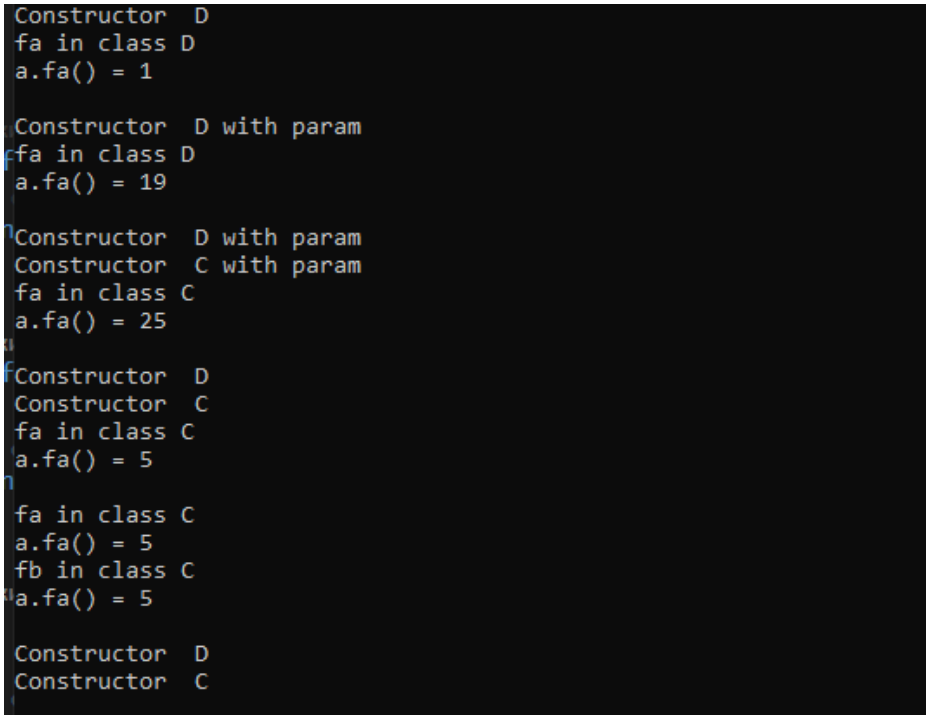
Лабораторная работа №5. Комбинирование через общих предков.



Текст программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab_5
{
    interface A  {
        int fa();
    }

    interface B : A
    {
        int fb();
    }

    class D : A
    {
        public D()
        {
            Console.WriteLine("Constructor  D");
            this.d = 1;
        }
        public D(int a)
```

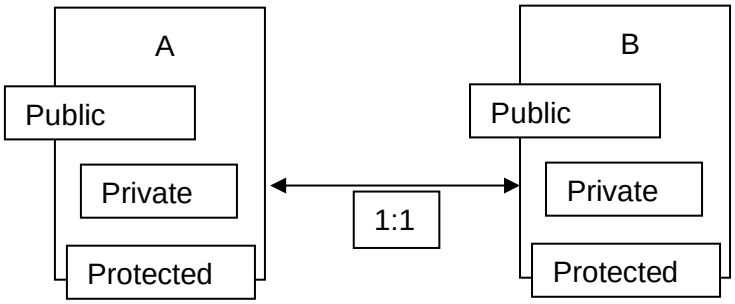
```
    {
        Console.WriteLine("Constructor  D with param");
        this.d = a;
    }
    public virtual int fa()
    {
        Console.WriteLine("fa in class D");
        return d;
    }
    protected int d { set; get; }
}

class C : D, B {
    public C()
    {
        Console.WriteLine("Constructor  C");
        this.c = 5;
    }
    public C(int a) :base(a)
    {
        Console.WriteLine("Constructor  C with param");
        this.c = a;
    }
    public override int fa()
    {
        Console.WriteLine("fa in class C");
        return c;
    }
    public int fb()
    {
        Console.WriteLine("fb in class C");
        return c;
    }
    protected int c { set; get; }
}

class Program
{
    static void Main(string[] args)
    {
        A a = null;
        a = new D();
        Console.WriteLine("a.fa() = {0}", a.fa());
        Console.WriteLine();
        a = new D(19);
        Console.WriteLine("a.fa() = {0}", a.fa());
        Console.WriteLine();
        a = new C(25);
        Console.WriteLine("a.fa() = {0}", a.fa());
        Console.WriteLine();
        a = new C();
        Console.WriteLine("a.fa() = {0}", a.fa());
        Console.WriteLine();

        Console.WriteLine("a.fa() = {0}", ((C)a).fa());
        Console.WriteLine("a.fa() = {0}", ((C)a).fb());
        Console.WriteLine();

        C c = new C();

        Console.ReadKey();
    }
}
                                                    }
```


Constructor D
fa in class D
a.fa() = 1
Constructor D with param
fa in class D
a.fa() = 19
Constructor D with param
Constructor C with param
fa in class C
a.fa() = 25
Constructor D
Constructor C
fa in class C
a.fa() = 5
fa in class C
a.fa() = 5
fb in class C
a.fa() = 5
Constructor D
Constructor C

Лабораторная работа №6. Ассоциация (один к одному, один ко многим).

Ассоциация - это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа.

Ассоциация один к одному – это когда один класс включает в себя другой класс в качестве одного из полей. Ассоциация описывается словом «имеет». Автомобиль имеет двигатель. Вполне естественно, что он не будет являться наследником двигателя (хотя такая архитектура тоже возможна в некоторых ситуациях).

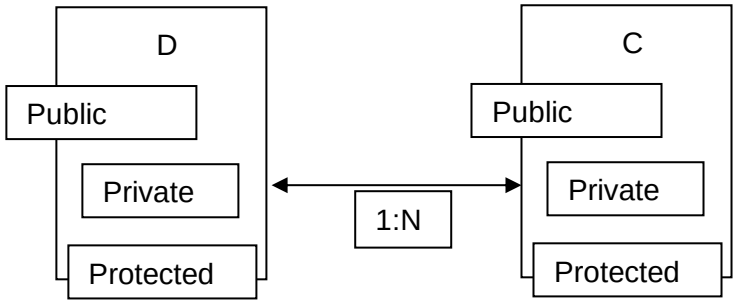


Один к одному

Текст программы:

```
using System;

namespace lab6_1
{
    class A
    {
        public A() { Console.WriteLine("Constructor A"); }
        public B b { set; get; }
        public int fa() { return 66; }
    }
    class B
    {
        public B() { Console.WriteLine("Constructor B"); }
        public A a { set; get; }
        public int fb() { return 666; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            A a = new A();
            B b = new B();

            b.a = a;
            a.b = b;

            Console.WriteLine("a.b.fb() = {0}", a.b.fb());
            Console.WriteLine("b.a.fa() = {0}", b.a.fa());
            Console.ReadKey();
        }
    }
}
```

```
Constructor A
Constructor B
a.b.fb() = 666
b.a.fa() = 66
```



Один ко многим

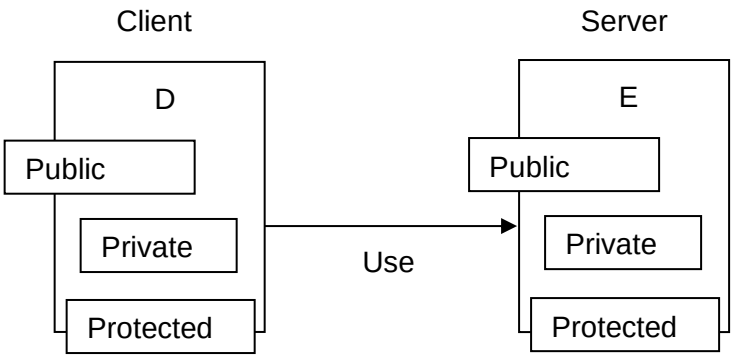
Текст программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace lab6_2
{
    class D
    {
        public D()
        {
            this.c = new C[N];
            Console.WriteLine("Constructor D");
        }
        public D(int n)
        {
            this.N = n;
            this.c = new C[N];
            Console.WriteLine("Constructor D for N");
        }
        public void setC(C c)
        {
            if (size < N)
            {
                this.c[size] = c;
                size++;
            }
        }
        //метод, позволяющий просматривать последовательно объекты класса C, связанные с объектом
класса D
        public C getNext(int index)
        {
            if (index < size)
            {
                return c[index];//возвращаем ссылку
```

```
            }
            return null;
        }

        public int size = 0;
        private C[] c = null;
        private int N = 5;
    }

    class C
    {
        public C()
        {
            Console.WriteLine("Constructor C");
        }
        public C(D d)
        {
            d.setC(this);
            Console.WriteLine("Constructor C with atribute");
        }
        public int j() { return v; }
        private int v = 10;
        public D d { set; get; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            D d1 = new D();
            D d2 = new D(10);
            Console.WriteLine();

            C c1 = new C();
            d1.setC(c1);
            C c2 = new C();
            d1.setC(c2);
            Console.WriteLine();

            c1.d = d1;
            Console.WriteLine("c1.d = {0}", c1.d);
            Console.WriteLine();

            Console.WriteLine("c1.d.getNext(1) = {0}", c1.d.getNext(1));
            Console.WriteLine("d1.getNext().j() = {0}", d1.getNext(1).j());
            Console.WriteLine("d1.getNext().j() = {0}", d1.getNext(1).j() + 1);
            Console.WriteLine();

            C c3 = new C(d1);

            Console.ReadKey();
        }
    }
}
```

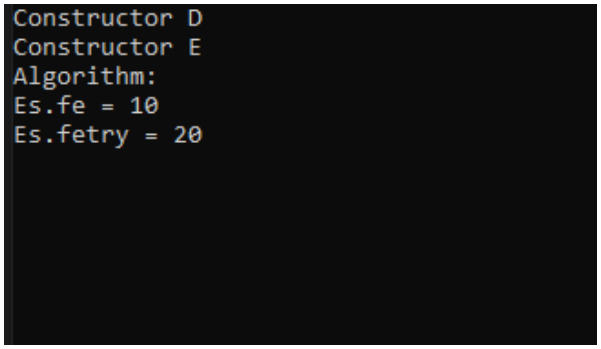
```
Constructor D
Constructor D for N

Constructor C
Constructor C

c1.d = lab6_2.D

c1.d.getNext(1) = lab6_2.C
d1.getNext().j() = 10
d1.getNext().j() = 11

Constructor C with atribute
```

Лабораторная работа №7. Использование.

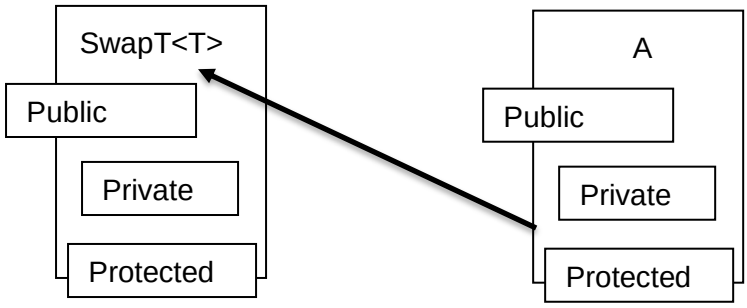


Текст программы:

```
using System;

namespace lab7
{
    class D //класс клиента
    {
        public D() { Console.WriteLine("Constructor D"); }
        public void fd(E e)
        {
            Console.WriteLine("Algorithm:");
            this.d = e.e + e.fe();
            Console.WriteLine("Es.fe = {0}", Es.fe()); //утилита
            Console.WriteLine("Es.fetry = {0}", Es.fetry()); //утилита
        }
        private int d { set; get; }
    }

    class E //класс сервера
    {
        public E() { Console.WriteLine("Constructor E"); }
        public int fe() { return 25; }
        public static int fs() { return 94; }
        public int e { set; get; }
    }

    Static class Es //класс утилиты
    {
        public static int fe() { return 10; }
        public static int fetry() { return 20; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            D d = new D(); //клиент
            E e = new E(); //сервер
            d.fd(e); // клиент обращается к ресурсам сервера
            Console.ReadKey();
        }
    }
}
```


Constructor D
Constructor E
Algorithm:
Es.fe = 10
Es.fetry = 20

Лабораторная работа №8. Конкретизация: параметров функции и метода, конструктора и атрибутов. Множественная конкретизация. Конкретизация с ограничениями (операция is).



Текст программы:

```
using System;

namespace lab8
{
    class A { }
    class B { }
    public interface IA
    {
        void F();
    }
    class D : IA
    {
        public void F() { Console.WriteLine("F() in class D"); }
    }
    class Swap
    {
        public void Fswap(ref A a, ref A b)
        {
            A temp = a;
            a = b;
            b = temp;
        }
    }

    class SwapT<T>where T:class //конкретизация с ограничениями (T классом)
    {
        //ref позволяет передавать параметры, иначе - делается копия
        public void Fswap(ref T a, ref T b) //конкретизация параметров функции
        {
            T temp = a;
            a = b;
            b = temp;
        }
    }

    class U<T> where T : class
    {
        public U(T t) //конкретизация конструктора
        {
            if (t is IA) //конкретизация с ограничениями
            {
                Console.WriteLine("It is IA");
                this.t = t; //конктетизация атрибука
            }
```

```
        else Console.WriteLine("It is not IA");
    }
    public void f1() { Console.WriteLine("Hi"); }
    public T t { set; get; }
    public void F()
    {
        Console.WriteLine(t is IA);
        Console.WriteLine(t);
        if (t is IA)
        {
            IA ia = (IA)t;
            ia.F();
        }
        else Console.WriteLine("False");
    }
}

class L<T1, T2> where T1:class //множественная конкретизация
                where T2:class
{
    public void f1(T1 t1, T2 t2)
    {
        Console.WriteLine("Hi T1, T2");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Swap swap = new Swap();
        A a1, a2 = null;
        a1 = new A();
        a2 = new A();
        Console.WriteLine();

        Console.WriteLine("a1 addr: {0}", a1.GetHashCode().ToString());
        Console.WriteLine("a2 addr: {0}", a2.GetHashCode().ToString());
        Console.WriteLine();

        swap.Fswap(ref a1, ref a2);
        Console.WriteLine("a1 addr: {0}", a1.GetHashCode().ToString());
        Console.WriteLine("a2 addr: {0}", a2.GetHashCode().ToString());
        Console.WriteLine();

        SwapT<A> swapt = new SwapT<A>();
        swapt.Fswap(ref a1, ref a2);
        Console.WriteLine("a1 addr: {0}", a1.GetHashCode().ToString());
        Console.WriteLine("a2 addr: {0}", a2.GetHashCode().ToString());
        Console.WriteLine();

        U<A> ua = new U<A>(a1);
        ua.F();
        Console.WriteLine();

        U<D> ud = new U<D>(new D());
        ud.F();
        D d1 = new D();
        Console.WriteLine("d1 addr: {0}", d1.GetHashCode().ToString());
        U<D> ud1 = new U<D>(d1);
        ud1.F();
        Console.WriteLine();

        L<D,A> l= new L<D,A>();
        l.f1(d1,a1);
        Console.ReadKey();
    }
```

```
    }
}
```

```
a1 addr: 58225482
a2 addr: 54267293

a1 addr: 54267293
a2 addr: 58225482

a1 addr: 58225482
a2 addr: 54267293

It is not IA
False

False

It is IA
True
lab8.D
F() in class D
d1 addr: 18643596
It is IA
True
lab8.D
F() in class D

Hi T1, T2
```

Лабораторная работа №9. Анонимные функции: сигнатура функций (delegate), Лямбда выражение(=>). Событие (Event).

Сигнатура функций (delegate)

Текст программы:

```
using System;

namespace lab9_1
{
    //делегат
    delegate int A(int x, int y);
    delegate void B();
    class Program
    {
        static void Main(string[] args)
        {
            B b = delegate () { Console.WriteLine("B b = delegate()"); };

            A a1 = new A(Add);
            A a = null;
            a = Add;
            Console.WriteLine();

            int result = a(20, 10);
            Console.WriteLine("Add(25, 7) = {0}",result);
            b();
            Console.WriteLine();

            Console.WriteLine("a = delegate (int x, int y) { return x % y; }");
            a = delegate (int x, int y) { return x % y; };
            Console.WriteLine("a(25, 7) = {0}", a(25, 7));
            b();
            Console.WriteLine();

            Console.WriteLine("a = delegate (int x, int y) { return x * y; }");
            a = delegate (int x, int y) { return x * y; };
            Console.WriteLine("a(25, 7) = {0}", a(25, 7));
            Console.WriteLine();

            Console.WriteLine("Delegate as poiner");
            A apoiner = null;
            apoiner = Add;
            apoiner += Multiply;
            Console.WriteLine("result {0}", apoiner.Invoke(37, 5));
            Console.WriteLine();

            apoiner -= Multiply;
            Console.WriteLine("apoiner -= Multiply = {0}", apoiner.Invoke(37, 5));
            apoiner += Multiply;
            Console.WriteLine("apoiner += Multiply = {0}", apoiner.Invoke(37, 5));
            apoiner -= Add;
            Console.WriteLine("apoiner -= Add = {0}", apoiner.Invoke(37, 5));
            apoiner += delegate (int x, int y) { return x * y; };
            Console.WriteLine("apoiner += delegate = {0}", apoiner.Invoke(37, 6));
            Console.WriteLine();

            Console.WriteLine("Delegate as parametr:");
            Multicast(15, 45, apoiner);

            Console.ReadKey();

        }
```

```
        private static int Add(int x, int y)
        {
            return x + y;
        }
        private static int Multiply(int x, int y)
        {
            return x * y;
        }
        public static void Multicast(int x, int y, A apoiner)
        {
            apoiner.Invoke(x, y);
        }

    }
}
```

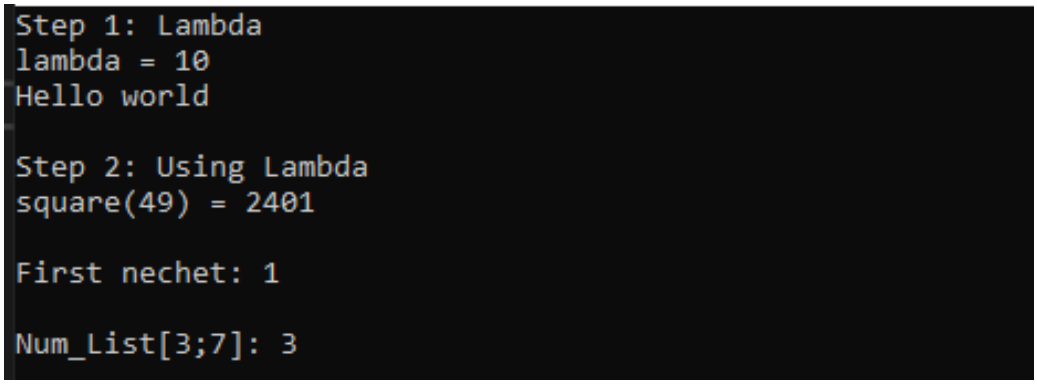
```
Add(25, 7) = 30
B b = delegate()

a = delegate (int x, int y) { return x % y; }
a(25, 7) = 4
B b = delegate()

a = delegate (int x, int y) { return x * y; }
a(25, 7) = 175

Delegate as poiner
result 185

apoiner -= Multiply = 42
apoiner += Multiply = 185
apoiner -= Add = 185
apoiner += delegate = 222

Delegate as parametr:
```

Лямбда выражение (=>)

Текст программы:

```
using System;
using System.Collections.Generic;

namespace lab9_2
{
    delegate int Lambda(int x, int y);
    delegate void Lambda_OP();
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Step 1: Lambda");
            //Lambda lambda = null;
            Lambda lambda = (x, y) => { return x + y; };
            Console.WriteLine("lambda = {0}", lambda(3, 7));
            Lambda_OP print = () => Console.WriteLine("Hello world");
            print();
            Console.WriteLine();

            Console.WriteLine("Step 2: Using Lambda");

            Func<int, int> square = (s) => s * s;
            Console.WriteLine("square(25) = {0}", square(49));
            Console.WriteLine();

            List<int> elements = new List<int>() {1, 2, 3, 4, 5, 6, 7, 8, 9};
            int Num = elements.Find(x => x % 2 != 0);
```

```
            Console.WriteLine("First nechet: {0}", Num);
            Console.WriteLine();

            int Num_List = elements.Find(x => (x >= 3 ) && (x <= 7));
            Console.WriteLine("Num_List[3;7]: {0}", Num_List);
            Console.ReadKey();
        }
    }
}
```



Событие (Event)

Текст программы:

```
using System;

namespace lab9_3
{
    public class Message : EventArgs
    {
        public string message { set; get; }
        public Message(string message) : base() { this.message = message; }
    }
    public class Publisher
    {
        public delegate void PublisherEventHandler(Message message);
        public delegate void EventHandler(Object sender, EventArgs args);
        public event PublisherEventHandler Changed;
        public void Ewraping(PublisherEventHandler Change)
        {
            Changed(new Message("Ewraping"));
        }
        public Publisher() { }
        public void EventForPublisher(Message message)
        {
            Console.WriteLine("Event for all subscribers {0}", message.message);
            Changed(message);
        }

    }
    public class Subscribers
    {
        int QRC { set; get; }
        public Subscribers(int QRC){ this.QRC = QRC; }
        public void subscribe(Message message) { Console.WriteLine("subscribe = {0}, {1}",
this.QRC, message.message); }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Step 1: Create");
            Publisher publisher = new Publisher();
            Subscribers Subscriber1 = new Subscribers(1);
```

```
            Subscribers Subscriber2 = new Subscribers(2);
            Subscribers Subscribern = new Subscribers(521);
            Console.WriteLine("Step 2: Sub");
            publisher.Changed += Subscriber1.subscribe;
            publisher.Changed += Subscriber2.subscribe;
            publisher.Changed += Subscribern.subscribe;
            Console.WriteLine("Step 3: Event");
            publisher.EventForPublisher(new Message("New book is ready!"));
            Console.ReadKey();
        }
    }
}
```

```
Step 1: Create
Step 2: Sub
Step 3: Event
Event for all subscribers New book is ready!
subscribe = 1, New book is ready!
subscribe = 2, New book is ready!
subscribe = 521, New book is ready!
```