**Перегрузка конструкторов, функций, операторов и операций**

Код программы:

using System;

namespace oop0

{

class Reestr

{

public string surname;

public int number;

public int yearOfPublishing;

public int time;

public Reestr()

{

surname = "IVAN";

number = 0;

yearOfPublishing = 2020;

time = 60;

}

public Reestr(string surname, int number, int yearOfPublishing, int time)

{

this.surname = surname;

this.number = number;

this.yearOfPublishing = yearOfPublishing;

this.time = time;

}

public Reestr(string surname, int number, int time)

{

this.surname = surname;

this.number = number;

yearOfPublishing = 2020;

this.time = time;

}

public void getInfo()

{

Console.WriteLine($"FIO: {surname}; NUM: {number}; YEAR: {yearOfPublishing}; TIME: {time}");

}

public void getInfo(string comment)

{

Console.WriteLine($"FIO: {surname}; NUM: {number}; YEAR: {yearOfPublishing}; TIME: {time}; COMMENT: {comment}");

}

public static Reestr operator +(Reestr person1, Reestr person2)

{

Reestr tempReestr = new Reestr();

tempReestr.time = person1.time + person2.time;

return tempReestr;

}

public string showTime()

{

return $"TIME: {this.time}";

}

public string showTime(int time)

{

return $"NEWTIME: {this.time + time}";

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("constr overloading");

Reestr Vasilyev = new Reestr("Васильев", 1, 2001, 30);

Reestr Yakubyshev = new Reestr("Якубышев", 2, 40); // перегрузка конструктора

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine("operation overloading");

Vasilyev.getInfo();

Yakubyshev.getInfo("Уволен"); // перегрузка операции

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine("operator overloading");

Console.WriteLine((Vasilyev + Yakubyshev).time); // перегрузка оператора

Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Console.WriteLine("function overloading");

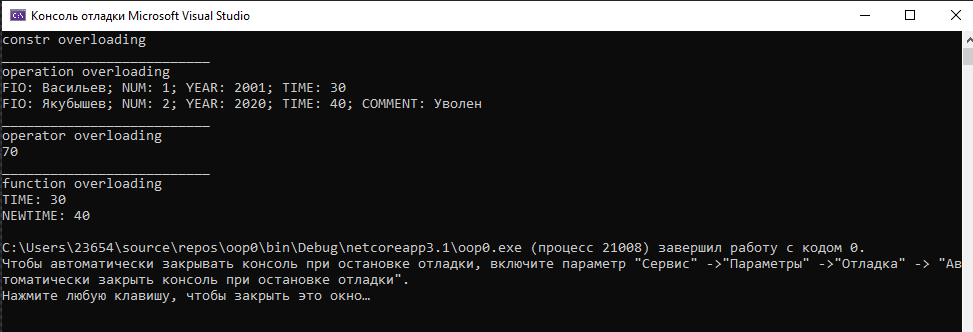
Console.WriteLine(Vasilyev.showTime());

Console.WriteLine(Vasilyev.showTime(10)); // перегрузка функции

}

}

}



**Вывод:**перегрузка конструкторов, функций, операторов и операций позволяет определить структурный полиморфизм, когда с данным именем существуют несколько функций с разным поведением, а также с различным списком аргументов, который определит, какая функция будет вызываться.

**Лабораторная работа №1. Агрегация по ссылке**

A

public

private

J

B

C

public

public

public

private

private

private

K

F

E

D

public

public

public

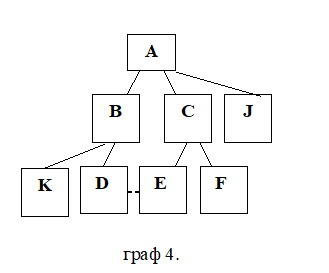
public

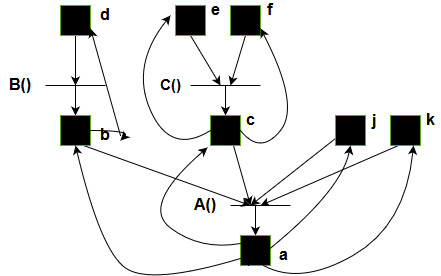
private

private

private

private





using System;

namespace oop2

{

class A

{

private B b = null;

private C c = null;

private J j = null;

public A(B b, C c, J j)

{

this.b = b;

this.c = c;

this.j = j;

c.cq = 4;

}

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 J jA

{

set { Console.WriteLine("set j"); j = value; }

get { Console.Write("get J ->"); return j; }

}

public C cA

{

set { Console.WriteLine("set c"); c = value; }

get { Console.Write("get C ->"); return c; }

}

}

class B

{

private K k = null;

private D d = null;

public B(K k, D d)

{

this.k = k;

this.d = d;

}

public void mB()

{

Console.WriteLine("Method of B");

}

public K kA

{

set { Console.WriteLine("set k"); k = value; }

get { Console.Write("get K ->"); return k; }

}

public D dA

{

set { Console.WriteLine("set d"); d = value; }

get { Console.Write("get D ->"); return d; }

}

}

class C

{

private E e = null;

private F f = null;

public C(E e, F f)

{

this.e = e;

this.f = f;

}

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; }

}

}

class J

{

public J() { }

public void mJ()

{

Console.WriteLine("Method of J");

}

}

class K

{

public K() { }

public void mK()

{

Console.WriteLine("Method of K");

}

}

class F

{

public F() { }

public void mF()

{

Console.WriteLine("Method of F");

}

}

class D

{

public D() { }

public void mD()

{

Console.WriteLine("Method of D");

}

}

class E

{

public E() { }

public void mE()

{

Console.WriteLine("Method of E");

}

}

class Program

{

static void Main(string[] args)

{

D d = new D();

E e = new E();

F f = new F();

K k = new K();

J j = new J();

C c = new C(e, f);

B b = new B(k, d);

A a = new A(b, c, j);

Console.WriteLine("передача по ссылке:");

a.mA();

a.cA.mC();

a.jA.mJ();

a.bA.mB();

a.cA.eA.mE();

a.cA.fA.mF();

a.bA.dA.mD();

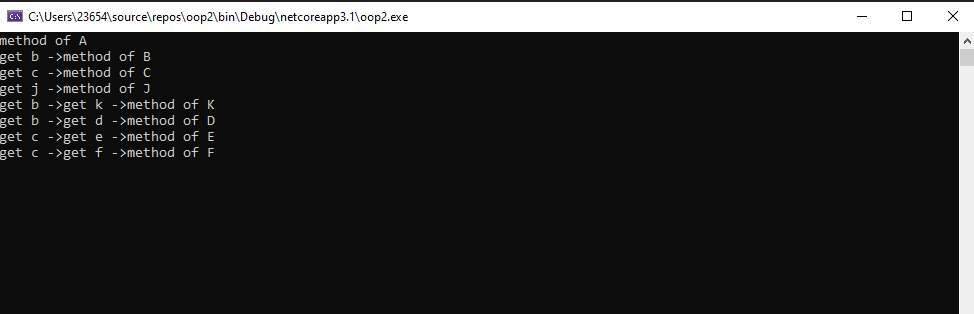
a.bA.kA.mK();

Console.ReadKey();

}

}

}



**Вывод: Иерархия целое часть** Объекты всех классов существуют независимо друг от друга. Связывание объектов происходит  с помощью конструктора.Например, *b, c, j, k —*параметры для конструктора класса A; *d* — для конструктора класса B; *e, f —*для конструктора класса C. 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 это объединение объектов в одну систему

**Агрегация по вложению.**

**A**

**B**

**C**

**J**

**K**

**E**

**F**

**b**

**c**

**j**

**k**

**e**

**f**

public

public

public

public

private

private

private

private

private

public

public

public

private

private

using System;

namespace oop2

{

class Program

{

class A

{

public A() { c.c1 = 4; }

public class B

{

public B() { }

public class K

{

public K() { }

public void mK() { Console.WriteLine("method of K"); }

}

public class D

{

public D() { }

public void mD() { Console.WriteLine("method of D"); }

}

public void mB() { Console.WriteLine(" method of B"); }

public K kA { get { Console.Write("get k -> "); return k; } }

public D dA { get { Console.Write("get d -> "); return d; } }

private K k = new K();

private D d = new D();

}

public class C

{

public C() { this.c1 = 10; }

public class E

{

public E() { }

public void mE() { Console.WriteLine(" method of E"); }

}

public class F

{

public F() { }

public void mF() { Console.WriteLine(" method of F"); }

}

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; } }

private E e = new E();

private F f = new F();

public int c1 { set; get; }

}

public class J

{

public 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();

}

static void Main(string[] args)

{

A a = new A();

a.mA();

a.bA.mB();

a.cA.mC();

a.jA.mJ();

a.bA.kA.mK();

a.bA.dA.mD();

a.cA.eA.mE();

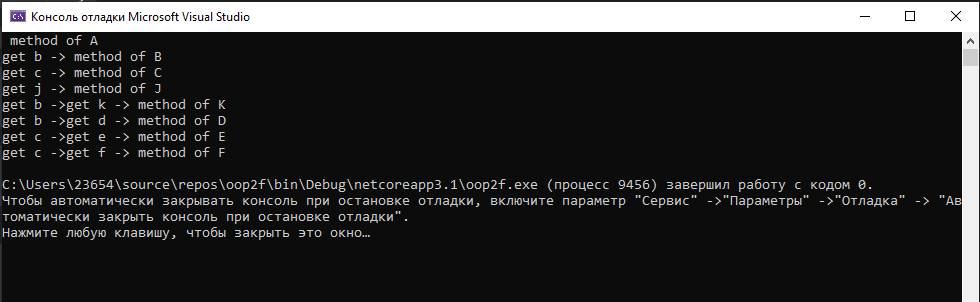
a.cA.fA.mF();

Console.ReadKey();

}

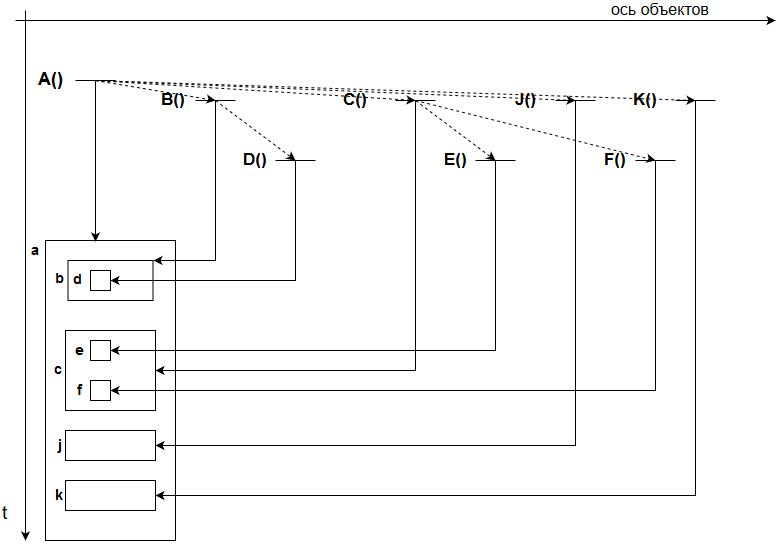
}

}



**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. При агрегации по вложению целый объект создается автоматически без необходимости описания каждого этапа сборки.

**Агрегация по значению**



**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab2\_Agr\_po\_Znach

{

class A

{

public A() { c.cq = 6; }

~A() { }

public void MethA() { Console.WriteLine("Method 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; }

}

B b = new B();

C c = new C();

J j = new J();

}

class B

{

public B() { }

~B() { }

public void MethB() { Console.WriteLine("Method B"); }

public K kA

{

set { Console.WriteLine("set k"); k = value; }

get { Console.Write("get k ->"); return k; }

}

K k = new K();

}

class C

{

public C() { this.cq = 6; }

~C() { }

public void MethC() { Console.WriteLine("Method 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 int cq { get; set; }

E e = new E();

F f = new F();

}

class K

{

public K() { }

~K() { }

public void MethK() { Console.WriteLine("Method K"); }

}

class E

{

public E() { }

~E() { }

public void MethE() { Console.WriteLine("Method E"); }

}

class F

{

public F() { }

~F() { }

public void MethF() { Console.WriteLine("Method F"); }

}

class J

{

public J() { }

~J() { }

public void MethJ() { Console.WriteLine("Method J"); }

}

class Program

{

static void Main(string[] args)

{

A a = new A();

a.MethA();

a.bA.MethB();

a.cA.MethC();

a.jA.MethJ();

a.bA.kA.MethK();

a.cA.eA.MethE();

a.cA.fA.MethF();

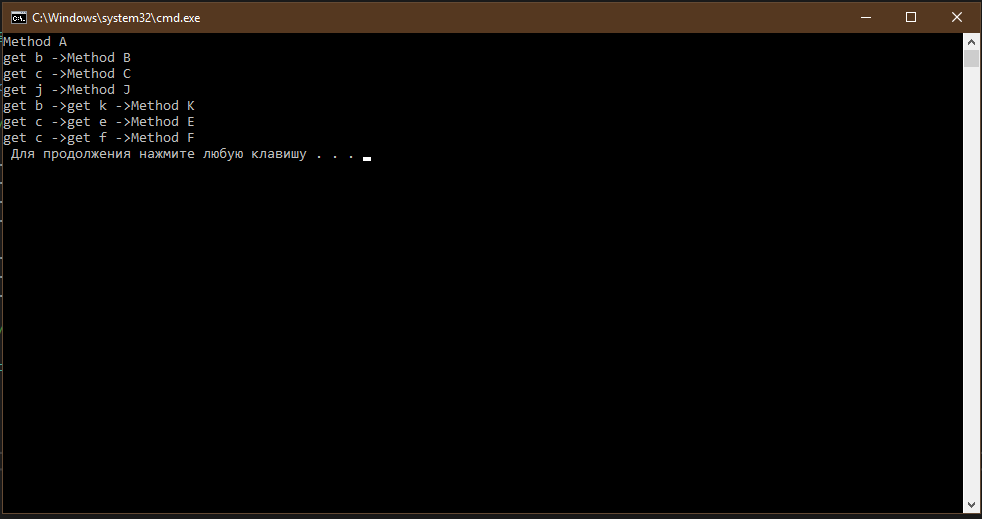
Console.ReadKey();

}

}

}

**Результат работы:**

****

**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 *а* класса А(первый класс по иерархии); эти части создаются только при вызове конструктора класса A, а уничтожаются — при вызове деструктора А. При агрегации по значению целый объект создается автоматически без необходимости описания каждого этапа сборки.

**Лаба №3. Принцип подстановки.**

using System;

using System.Runtime.InteropServices;

namespace oop3

{

class A

{

public A()

{

Console.WriteLine("object A created");

this.a = 1;

}

public virtual int fa() { return a + 1; }

protected int a { set; get; }

}

class C : A

{

public C()

{

this.a = 20;

Console.WriteLine("object C created");

}

//замещение

public override int fa() { return a + 10; }

}

class E : C

{

public E()

{

this.a = 300;

Console.WriteLine("object E created");

}

//замещение

public override int fa() { return a + 100; }

}

class Program

{

static void Main(string[] args)

{

A a = new A();

Console.WriteLine($"{a.fa()}");

//подстановка

a = new C();

Console.WriteLine($"{a.fa()}");

//подстановка

a = new E();

Console.WriteLine($"{a.fa()}");

if (a.GetType() == typeof(C))

{

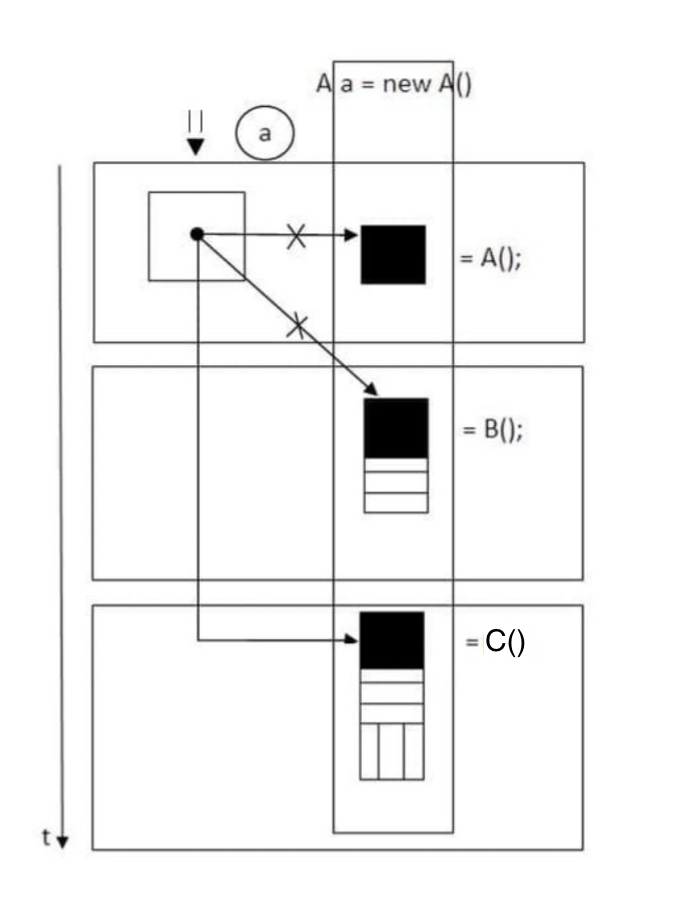
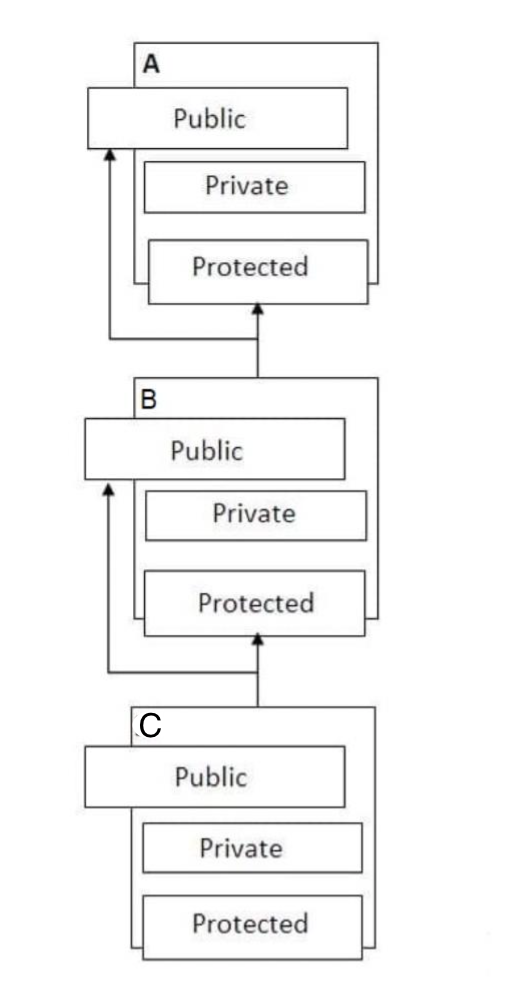
Console.WriteLine("help");

}

}

}

}

Лабораторная работа №3 “Принцип подстановки”

using System;

namespace Laba3

{

public class A

{

public A()

{

Console.WriteLine("Method of A");

this.varA = 55;

}

~A()

{

Console.WriteLine("Destructor of A");

}

public virtual int Function()

{

Console.WriteLine("Func of class A");

return this.varA + 445;

}

protected int varA { get; set; }

}

public class B : A

{

public B()

{

Console.WriteLine("Method of B");

this.varA = 78;

}

~B()

{

Console.WriteLine("Destructor of B");

}

public override int Function()

{

Console.WriteLine("Func of class B");

return this.varA + 78;

}

}

public class C : B

{

public C()

{

Console.WriteLine("Method of C");

this.varA = 4;

}

~C()

{

Console.WriteLine("Destructor of C");

}

public override int Function()

{

Console.WriteLine("Func of class C");

return (this.varA + 32);

}

}

class Program

{

static void Main(string[] args)

{

A a = new A();

Console.ReadKey();

Console.WriteLine("Function : {0}",a.Function());

a = new B();

Console.ReadKey();

Console.WriteLine("Function : {0}", a.Function());

a = new C();

Console.ReadKey();

Console.WriteLine("Function : {0}", a.Function()); //функция замещена

if (a.GetType() == typeof(B))

Console.WriteLine("(a.GetType() == typeof(B))"); //какой объект подставлен

else

{

Console.WriteLine("(a.GetType() != typeof(B))");

if (a.GetType() == typeof(C))

Console.WriteLine("(a.GetType() == typeof(C))");

else

Console.WriteLine("(a.GetType() != typeof(C))");

}

{

C c = new C();

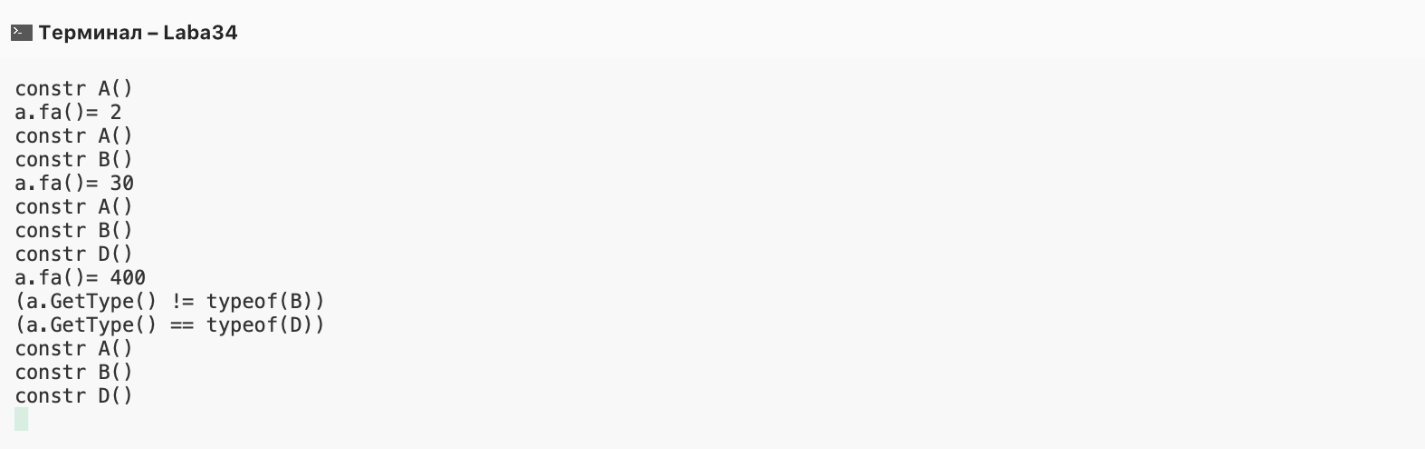
}

Console.ReadKey();

}

}

}



**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.

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

**Наследование: расширение, спецификация, специализация, конструирование**

**А**

**B**

**C**

**J**

**K**

**E**

**F**

public

public

public

public

public

public

public

private

private

private

private

private

private

private

public

private

**D**

**V**

public

private

**Exp**

**Sa**

**Sa**

**Cn**

**Sp**

**Si**

**Ex – расширение B -> A**

**Sa- Спецификация: Abstract class C**

**Si-Спецификация : Interface J**

**Sp-Специализация K->B**

**Cn – конструирование V->K**

*Рис. 1 Наследование: расширение, спецификация, специализация, конструирование.*

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace ConsoleApp1\_Lab\_4\_

{

class A

{

public A()

{

Console.WriteLine("constr A()");

this.a = 1;

}

~A()

{

Console.WriteLine(" destr A()");

Thread.Sleep(80000);

}

public virtual int fa() { Console.WriteLine("class A fa()"); return a + 1; }

public int Aa

{

set { Console.Write("set"); }

get { Console.Write("get"); return a; }

}

protected int a = 1;

}

class B : A

{

public B()

{

Console.WriteLine("constr B()");

this.a = 20;

this.b = 10;

this.b1 = -1;

}

~B()

{

Console.WriteLine("destr B()");

}

public override int fa()

{

Console.WriteLine("class B fa()");

base.fa();

a = a + 10;

return a;

}

public int fb()

{

Console.WriteLine("class B fb()");

return a + b + 10;

}

protected int b { set; get; }

public int b1 { set; get; }

}

class K : B

{

public K()

{

Console.WriteLine("constr K()");

this.a = 300;

}

public override int fa() { Console.WriteLine("class K fa()"); return a + 100; }

~K()

{

Console.WriteLine("destr K()");

}

}

abstract class C

{

abstract public int fc();

public void print() { Console.WriteLine("class C print"); }

public int Cc

{

set { Console.Write(""); }

get { Console.Write("get"); return c; }

}

protected int c = 1;

}

class E : C

{

public E() { this.c = 22; }

public override int fc()

{

Console.WriteLine("class E fc()");

return c \* 5;

}

}

class F : C

{

public F() { this.c = 22; }

public override int fc()

{

Console.WriteLine("class F fc()");

return c \* 50;

}

}

interface J

{

int fj\_1();

int fj\_2();

}

class D : J

{

public D() { }

public int fj\_1() { return 250; }

public int fj\_2() { return 500; }

}

class V : K

{

public V() { }

public override int fa() { Console.WriteLine("not use"); return 0; }

}

class Program

{

static void Main(string[] args)

{

A a = new A();

Console.ReadKey();

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

a = new B();

Console.ReadKey();

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

/\*a = new K();

Console.ReadKey();

Console.WriteLine("a.fa()= {0}", a.fa());\*/

if (a.GetType() == typeof(B))

Console.WriteLine("(a.GetType() == typeof(B))");

else

{

Console.WriteLine("(a.GetType() != typeof(B))");

if (a.GetType() == typeof(K))

Console.WriteLine("(a.GetType() == typeof(K))");

else

Console.WriteLine("(a.GetType() != typeof(K))");

}

{

// K k = new K();

}

//Thread.Sleep(10000);

Console.ReadKey();

Console.WriteLine("Expansion");

Console.WriteLine("((B)a).fb () = {0}", ((B)a).fb()); // Расширение по функции

Console.WriteLine("((B)a).b1 () = {0}", ((B)a).b1); // Расширение по аргументу

Console.ReadKey();

Console.WriteLine("Specification : abstract class");

C c = null;

c = new E();

c.Cc = 455;

c.print();

Console.WriteLine("c.fc()={0}", c.fc());

Console.ReadKey();

c = new F();

c.Cc = 455;

c.print();

Console.WriteLine("c.fc()={0}", c.fc());

Console.ReadKey();

Console.WriteLine("Specification : interface");

J j = null;

j = new D();

Console.WriteLine("j.fj\_1() ={0}", j.fj\_1());

Console.WriteLine("j.fj\_2() ={0}", j.fj\_2());

Console.ReadKey();

Console.WriteLine("Specialization");

a = new K();

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

Console.ReadKey();

Console.WriteLine("Construction");

a = new V();

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

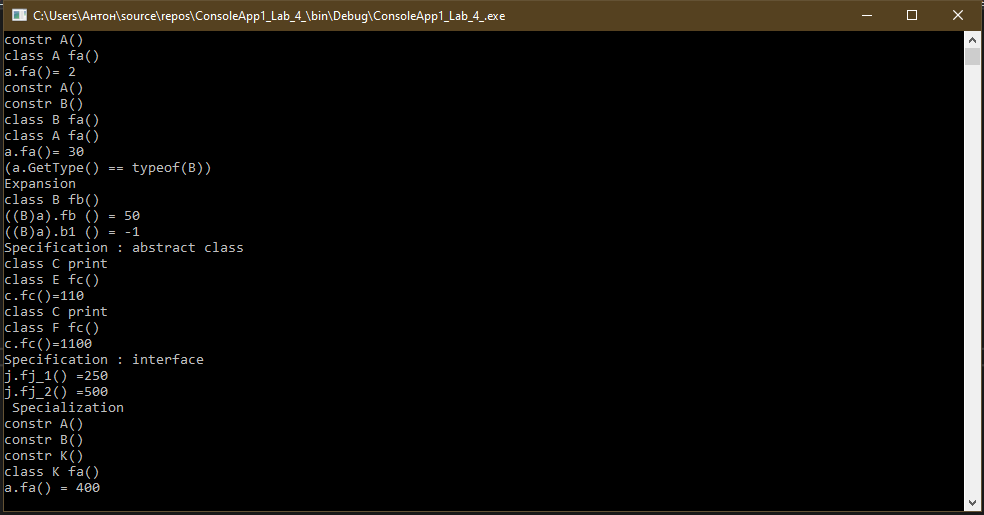
Console.ReadKey();

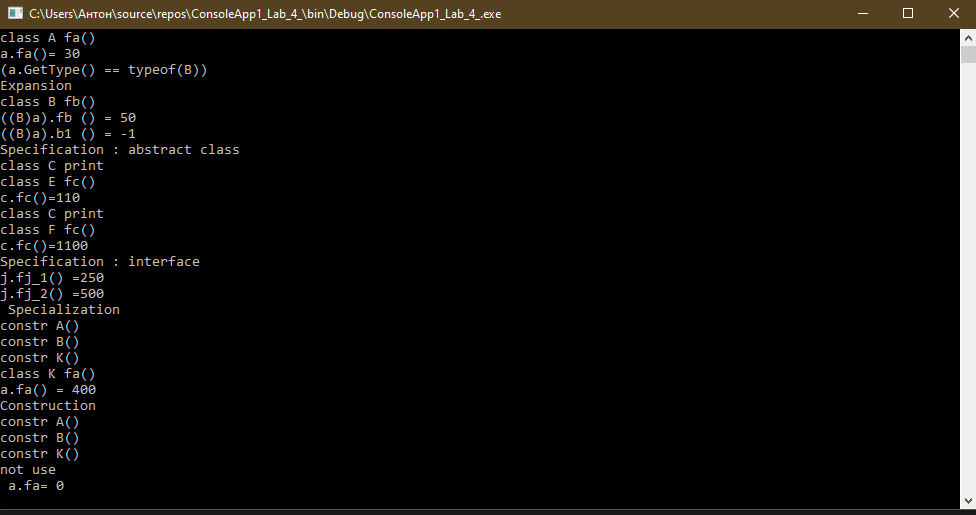
}

}

}

**Результат работы :**

****

****

**Вывод:**

**Наследование: комбинирование.**

**А**

**B**

**C**

**J**

**K**

**E**

**F**

public

public

public

public

public

public

public

private

private

private

private

private

private

private

*Рис 2. Наследование: комбинирование.*

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab\_4.\_2\_

{

class K

{

public K() { Console.WriteLine("constr K"); }

public int k { get; set; }

public virtual void fk() { Console.WriteLine("fk K"); }

}

class B:K

{

public B() { Console.WriteLine("constr B"); }

public int b { get; set; }

public virtual void fb() { Console.WriteLine("fb B"); }

public override void fk() { Console.WriteLine("Override fk B"); }

}

interface E

{

double fe();

}

interface F

{

double ff();

}

interface C:E,F

{

int fc();

}

interface J

{

int fj();

int fj1();

}

class A : B, C, J

{

public A() { Console.WriteLine("constr A"); }

public int a { get; set; }

public override void fb()

{

base.fb();

Console.WriteLine(" fb Override A");

Console.WriteLine(" Override fb ");

}

public override void fk() { Console.WriteLine("Override fk A"); }

public double fe() { return 56.25; }

public double ff() { return 112.5; }

public int fc() { return 255; }

public int fj() { return 510; }

public int fj1() { return 1020; }

}

class Program

{

static void Main(string[] args)

{

B b = new B();

b.b = 11;

b.fb();

b.fk();

Console.WriteLine("b.b={0}", b.b);

b = new A();

Console.ReadKey();

b.fb();

b.fk();

Console.ReadKey();

Console.WriteLine("Step 2 : Interface C:E,F ");

C c = null;

c = new A();

Console.WriteLine("c.fc()={0}", c.fc());

Console.WriteLine("c.fe()={0}, c.ff()={1}", c.fe(), c.ff());

Console.WriteLine("Step 3 : Interface J ");

J j = null;

j = new A();

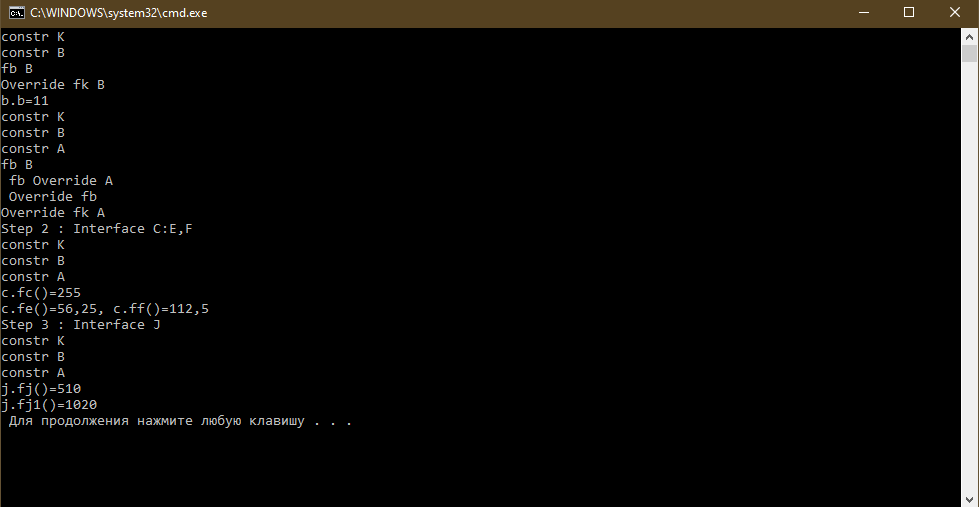
Console.WriteLine("j.fj()={0}", j.fj());

Console.WriteLine("j.fj1()={0}", j.fj1());

Console.ReadKey();

}

}

**Результат работы: **

**Вывод:**

**Лабораторная работа №5. Наследование: комбинирование через общих предков.**

**А**

**B**

**C**

**K**

public

public

public

public

private

private

private

private

*Рис 1. Наследование: комбинирование через общих предков.*

**Текст программы:**

using System;

namespace laba5\_oop

{

interface A

{

public void MethodOfA();

}

class B : A

{

public B()

{

Console.WriteLine("Конструктор B");

this.b = 1;

}

public B(int a)

{

this.b = a;

Console.WriteLine("Конструктор B с параметрами");

}

public virtual void MethodOfA()

{

Console.WriteLine("Это реализация метода интерфейса А классом B");

}

public virtual void Method()

{

Console.WriteLine("Это метод класса B");

}

protected int b { get; set; }

}

interface C : A

{

public void MethodOfC();

}

class D : B, C

{

public D()

{

this.d = 1;

Console.WriteLine("Конструктор D");

}

public D (int a):base(a)

{

this.d = a;

Console.WriteLine("Конструктор c параметрами D");

}

public override void MethodOfA()

{

Console.WriteLine("Это реализация метода интерфейса А классом D");

}

public void MethodOfC()

{

Console.WriteLine("Это реализация метода интерфейса C классом D");

}

public override void Method()

{

Console.WriteLine("Это метод класса D");

}

public int d { get; set; }

}

class Program

{

static void Main(string[] args)

{

B b = new B();

Console.WriteLine();

D d = new D();

d = new D(15);

Console.WriteLine();

b.MethodOfA();

Console.WriteLine();

b.Method(); //метод класса В реализованный интерфейсом А

Console.WriteLine();

d.MethodOfA(); //Это реализация метода интерфейса А классом D

Console.WriteLine();

d.MethodOfC(); //Это реализация метода интерфейса C классом D

Console.WriteLine();

d.Method();

Console.WriteLine();

Console.ReadKey();

}

}

}

**Результат программы:**

