¶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 С#**

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

9. АНОНИМНЫЕ ФУНКЦИИ: delegate, ЛЯМБДА ВЫРАЖЕНИЕ. Event

Работу выполнил:

8О-205Б ........................ \_\_\_\_\_\_\_\_\_ Вариант 6

Руководитель: \_\_\_\_\_\_\_/Семенов А.С.

Дата: \_\_\_ октября 2020

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

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab\_0

{

class Reestr //Перегрузка: конструктор, функции, операции, операторы.

{

public override string ToString()

{

return $"Фамилия: {surn}; Номер в списке: {numb.ToString()}; Год публикации: {publyear.ToString()}; Возраст: {age.ToString()}";

}

string surn = "---" ;

int numb = 0 ;

int publyear = 2020 ;

int age= 0 ;

public Reestr() //конструктор

{

Console.WriteLine("Constructor without parameters");

}

public Reestr(string surn, int numb, int publyear, int age)

{

this.surn = surn;

this.numb = numb;

this.publyear = publyear;

this.age = age;

Console.WriteLine("Constructor with parameters");

}

public void SurAge() //операция

{

Console.WriteLine("Operation without parameters");

Console.Write("{0}\t", surn);

Console.Write("{0}\t", age);

Console.Write("- Значение имени и возраста по умолчанию");

Console.WriteLine();

}

public void SurAge(int NoInd)

{

Console.WriteLine("Operation with parameter");

Console.Write("{0}\t", numb);

Console.WriteLine("{0}\t","- Номер в списке");

Console.WriteLine("");

}

public int Agediff() //функция

{

Console.WriteLine("Function without parameters");

return 2020 - this.publyear;

}

public int Agediff(int NoInd)

{

Console.WriteLine("Function with parameter");

return 2020 - this.publyear + NoInd;

}

public static int operator +(Reestr unknowperson, Reestr homskiy)//оператор

{

Console.WriteLine("Overload:");

return unknowperson.publyear + homskiy.publyear;

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Overloading constructor");

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

Reestr unknowperson = new Reestr();

Console.WriteLine(unknowperson.ToString());

Reestr homskiy = new Reestr("Хомский", 1, 1950, 55);

Console.WriteLine(homskiy.ToString());

Console.WriteLine();

Console.WriteLine("Overloading operation");

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

unknowperson.SurAge();

homskiy.SurAge(3);

Console.WriteLine("Overloading function");

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

Console.WriteLine(unknowperson.Agediff());

Console.WriteLine(homskiy.Agediff(30));

Console.WriteLine();

Console.WriteLine("Overloading operator");

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

Console.WriteLine( unknowperson + homskiy) ;

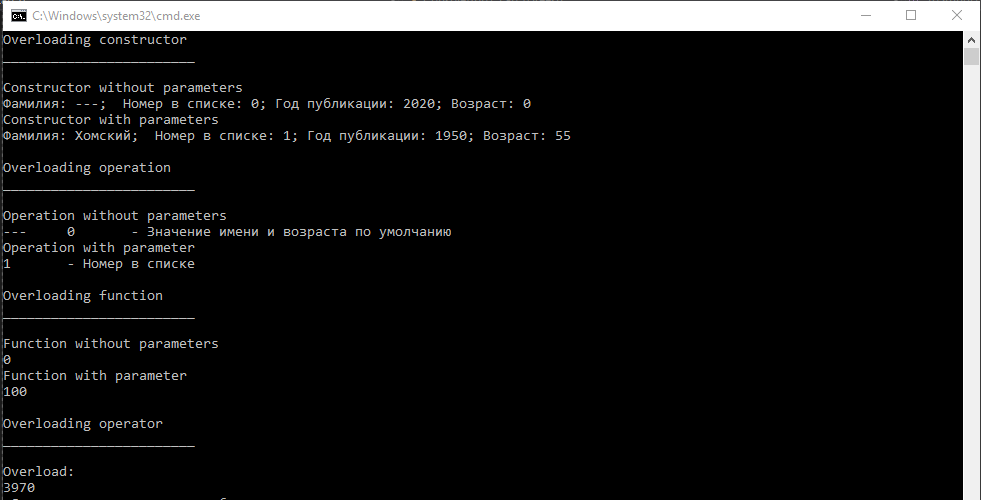
Console.ReadKey();

}

}

}

**Результат работы:**

****

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

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

**А**

**B**

**C**

**J**

**K**

**E**

**F**

public

public

public

public

public

public

public

private

private

private

private

private

private

private

***Рис.1 Агрегация по ссылке.***

**A**

**B**

**C**

**J**

**E**

**K**

**F**

**B(**

**C(**

**A(**

***Рис.2 Диаграмма классов: агрегация по ссылке.***

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab\_1

{

class A

{

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

{

this.b = b;

this.c = c;

this.j = j;

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 = null;

C c = null;

J j = null;

}

class B

{

public B(K k)

{

this.k = k;

}

~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 = null;

}

class C

{

public C(E e, F f)

{

this.e = e;

this.f = f;

}

~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 = null;

F f = null;

}

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)

{// Объекты, из которых будут созданы другие.

K k = new K();

E e = new E();

F f = new F();

J j = new J();

// передача вышестоящих объектов в виде параметров в конструкторы.

B b = new B(k);

C c = new C(e, f);

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

// вывод атрибута доступа.

Console.WriteLine("Атрибут доступа = {0} ",c.cq);

// передача по ссылке: проверка

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

a.MethA();

a.bA.MethB();

a.cA.MethC();

a.jA.MethJ();

a.bA.kA.MethK();

a.cA.eA.MethE();

a.cA.fA.MethF();

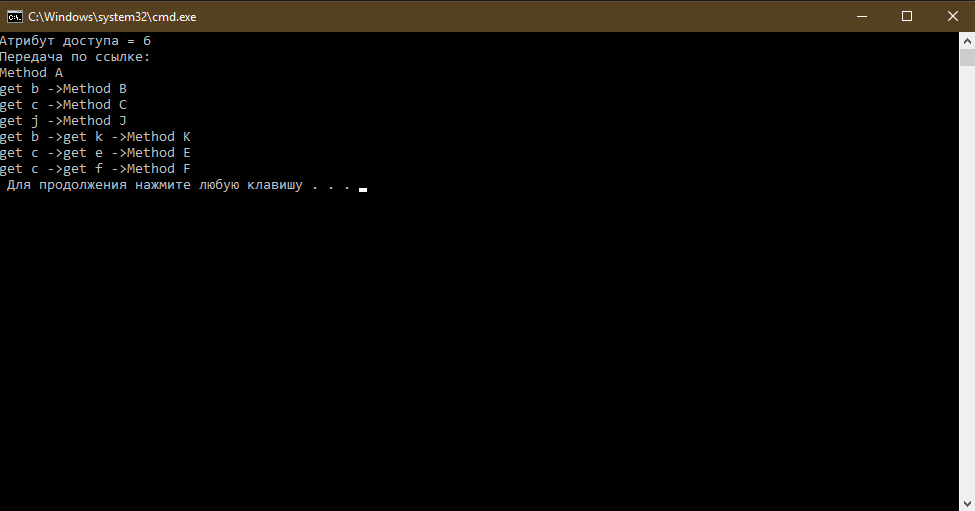
Console.ReadKey();

}

}

}

**Результат работы:**

****

**Вывод:** Объекты всех классов существуют независимо друг от друга. Связывание объектов происходит с помощью конструктора. Например, *b, c, j —* параметры для конструктора класса A; *k* — для конструктора класса B; *e, f —* для конструктора класса C. Объекты могут быть уничтожены по отдельности. Это не нарушит целостность структуры. Если удалить объект *a,* объекты *b, c, j* и т.д будут продолжать существовать и дальше. При агрегации по ссылке каждый этап сборки целого объекта описывается вручную, в отличие от агрегации по значению и вложению.

**Лабораторная работа №2. Агрегация по значению и вложению.**

**Агрегация по значению.**

**А**

**B**

**C**

**J**

**K**

**E**

public

public

public

public

private

private

private

private

**F**

public

public

private

private

public

private

**b**

**c**

**j**

**k**

**e**

**f**

***Рис1. Агрегация по значению.***

***Рис.2 Диаграмма классов: агрегация по значению.***

Ось объектов

t

**a**

**b**

**k**

**c**

**j**

**e**

**f**

**A(**

**B(**

**K(**

**C(**

**E(**

**F(**

**J(**

**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)

{

//C c = new C();

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.WriteLine(a.cA.cq);

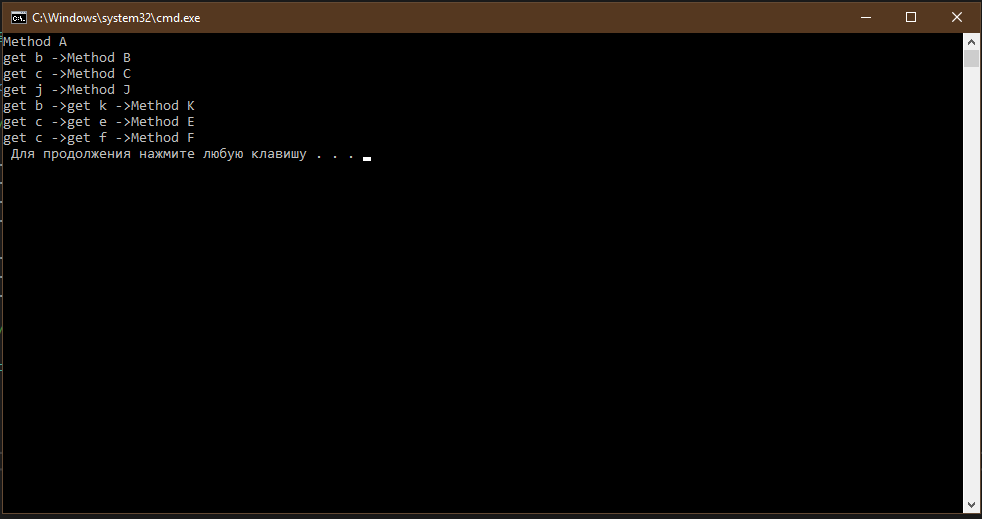
Console.ReadKey();

}

}

}

**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.

**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

***Рис.1 Диаграмма классов: агрегация по вложению.***

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab2\_Agr\_po\_Vlozh

{

class A

{

public A() { c.cq = 5; }

~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();

public class B // вложенный класс B в A

{

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();

public class K // вложенный класс K в B

{

public K() { }

~K() { }

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

}

}

public class C // вложенный класс C в A

{

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();

public class E // вложенный класс E в C

{

public E() { }

~E() { }

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

}

public class F // вложенный класс F в C

{

public F() { }

~F() { }

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

}

}

public class J // вложенный класс J в A

{

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();

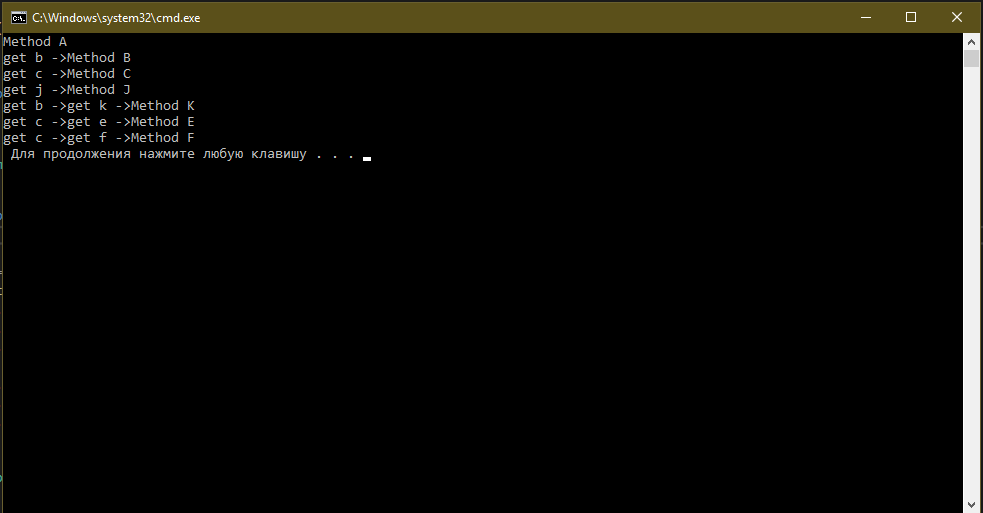
Console.ReadKey();

}

}

}

**Результат работы:**

****

**Вывод:** при агрегации 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.

**Лабораторная работа №3. Принцип подстановки.**

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab2\_Agr\_po\_Vlozh

{

class A

{

public A() { c.cq = 5; }

~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();

public class B // вложенный класс B в A

{

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();

public class K // вложенный класс K в B

{

public K() { }

~K() { }

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

}

}

public class C // вложенный класс C в A

{

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();

public class E // вложенный класс E в C

{

public E() { }

~E() { }

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

}

public class F // вложенный класс F в C

{

public F() { }

~F() { }

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

}

}

public class J // вложенный класс J в A

{

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();

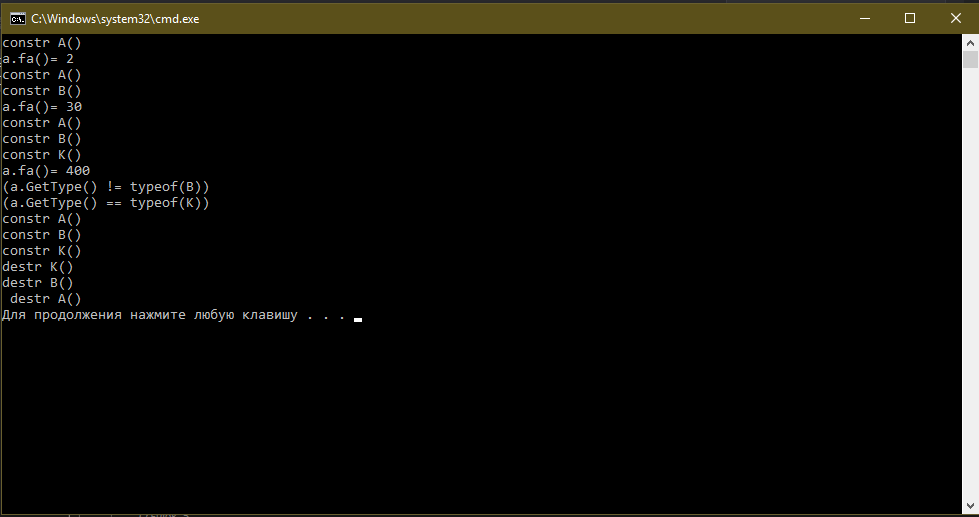
Console.ReadKey();

}

}

}

**Результат работы:**

****

**Вывод :** 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, в производном классе. Например, этот метод может быть переопределен любым наследующим его классом: модификатор override требуется изменения реализации унаследованного метода, свойства.

**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());

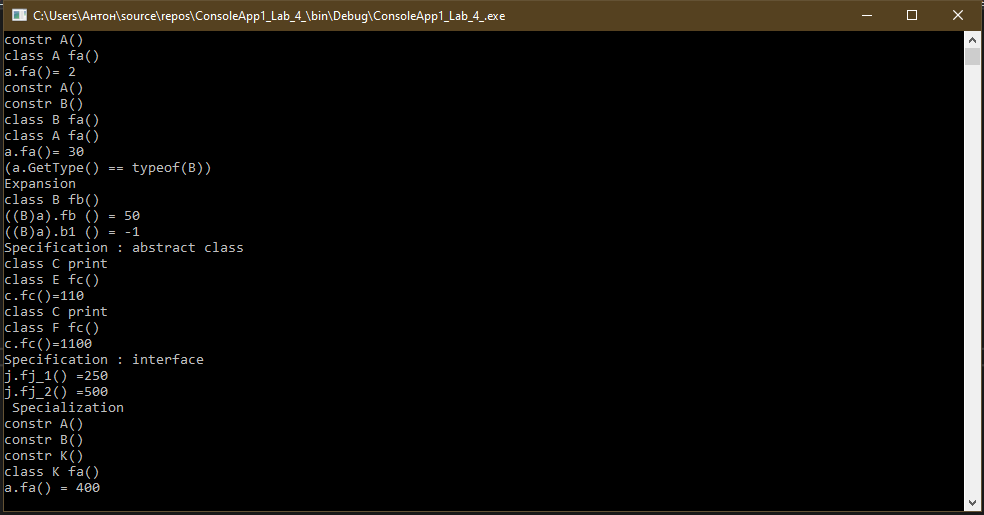
Console.ReadKey();

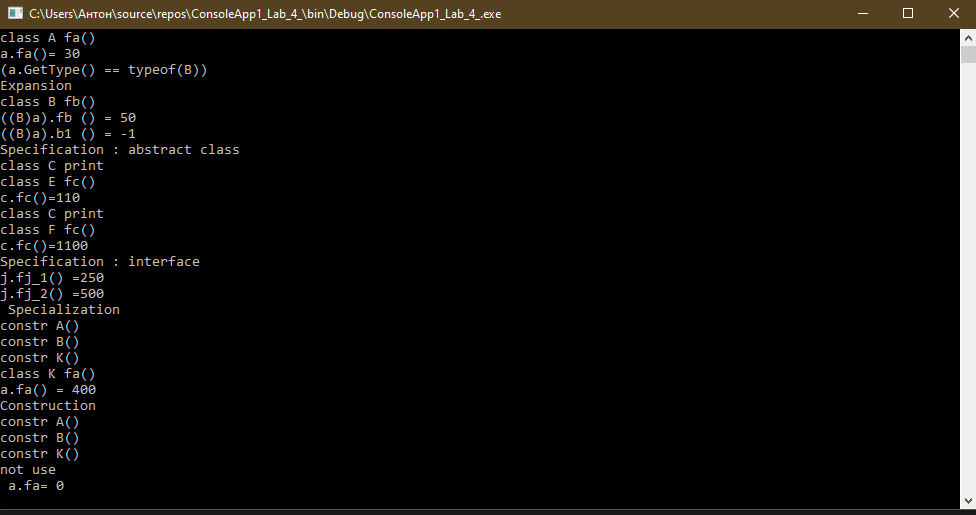
}

}

}

**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();

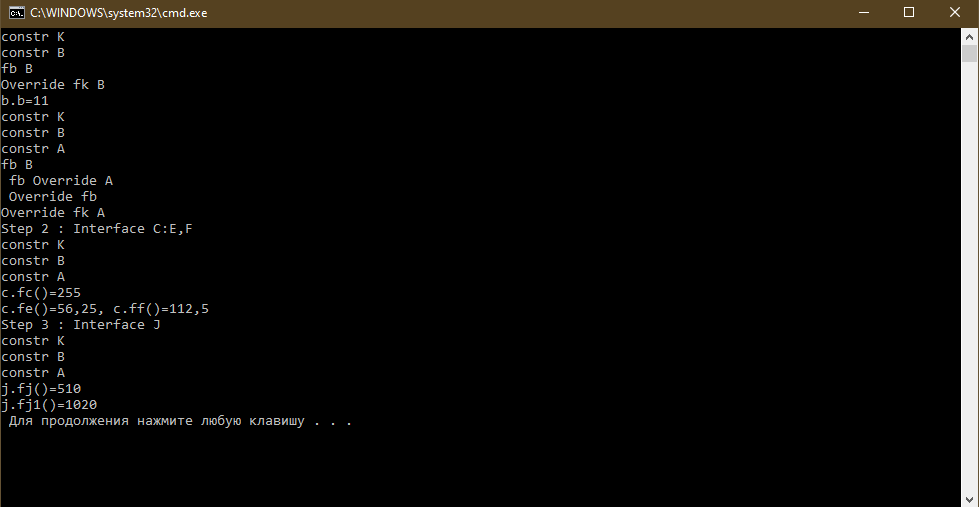
Console.WriteLine("j.fj()={0}", j.fj());

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;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1\_Lab\_5

{

interface K

{

int fk();

}

interface C:K

{

int fc();

int fc\_1();

}

class B:K

{

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

public int fk() { return 2020; }

public virtual int print()

{

Console.WriteLine("Hello !");

return 0;

}

protected int b {set;get;}

}

class A : B,C

{

public A() { this.a = 1; Console.WriteLine("Constr A"); }

public int fc() { return a + 2020; }

public int fc\_1() { return 2020 - a; }

public int fk() { return a; }

public override int print()

{

base.print();

Console.WriteLine("Hello, I'm here !");

return 0;

}

public int a { set; get; }

}

class Program

{

static void Main(string[] args)

{

K k = null;

k = new A();

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

Console.WriteLine("((C)k).fc()={0}", ((A)k).fc());

Console.WriteLine("((C)k).fc\_1()={0}", ((A)k).fc\_1());

((A)k).print();

A a = new A();

a.print();

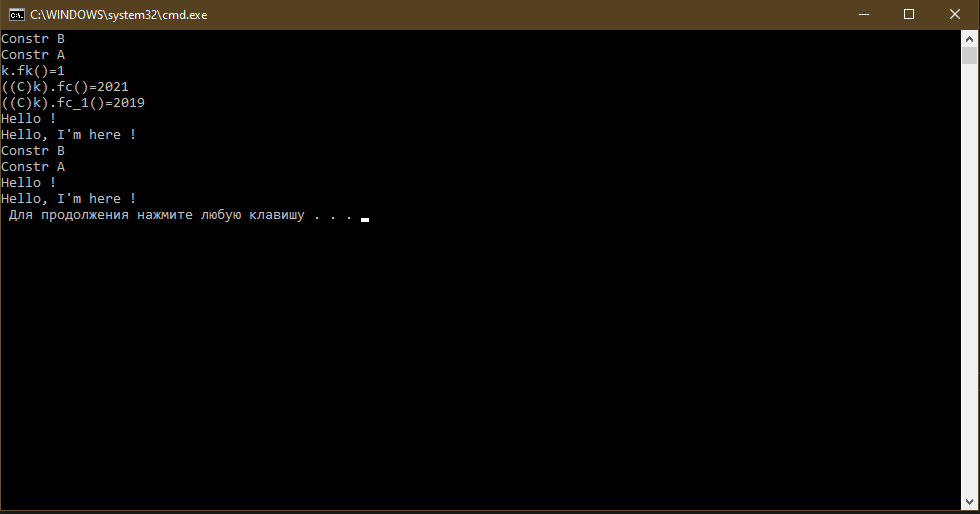
Console.ReadKey();

}

}

}

**Результат программы:**

****

**Вывод:**