Московский авиационный институт

(национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

Кафедра вычислительной математики и программирования

**Лабораторная работа 4 по курсу ОOП:**

**основы программирования на языке С#**

НАСЛЕДОВАНИЕ. РАСШИРЕНИЕ, СПЕЦИФИКАЦИЯ, СПЕЦИАЛИЗАЦИЯ

Работу выполнил:

Березнев Никита Вадимович

Группа: М8О-203Б-21

Подпись: \_\_\_\_\_\_\_\_

Дата: 29 октября 2022 г.

**4.1 Наследование. Расширение, спецификация, специализация**



**Код программы:**

using System;

namespace CSharpLab4\_1

{

public interface A

{

void mA();

int fA();

}

public class C : A

{

public C()

{

this.c = 1;

this.c1 = 4;

}

~C() { Console.WriteLine("Диструктор С"); }

public void mA() { Console.WriteLine("метод mA интерфейса A"); }

public virtual int fA()

{

this.c += 1;

return 0;

}

public virtual int fC() { return 1; }

public virtual void mC() { Console.WriteLine("метод mС класса С"); }

public virtual int S() { return c1 \* 2; }

protected int c { get; set; }

public int c1 { get; set; }

}

public class D : C

{

public D()

{

this.d1 = 1;

this.d2 = 2;

}

~D() { }

public override int fC()

{

Console.WriteLine("Method of abstract class С in class D.");

return this.d1 \* this.d2 + base.fC();

}

public override void mC()

{

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

}

public int fD()

{

Console.WriteLine("Method of D");

return this.d1 + this.d2;

}

public int d1 { get; set; }

public int d2 { get; set; }

}

public class E : C

{

public E()

{

this.e1 = 1;

this.e2 = 2;

}

~E() { }

public virtual int S() { return c1 + 69; }

public override void mC()

{

Console.WriteLine("метод mС класса E");

}

public int e1 { get; set; }

public int e2 { get; set; }

}

public class F : C

{

public F()

{

this.f1 = 1;

this.f2 = 2;

}

~F() { }

public override int fC()

{

Console.WriteLine("Method of abstract class С in class E.");

return this.f1 \* this.f2 + base.fC();

}

public override void mC()

{

Console.WriteLine("метод mС класса E");

}

public int fF()

{

Console.WriteLine("Method of E");

return this.f1 + this.f2;

}

public int f1 { get; set; }

public int f2 { get; set; }

}

public abstract class J : A

{

public J()

{

this.j1 = 10;

this.j2 = 20;

}

~J() { }

public void l() { Console.WriteLine(+20); }

public void mA() { Console.WriteLine("Метод интерфейса A, реализованный в классе J"); }

public int fA() { return j1 + j2; }

public abstract void mJ();

protected virtual int pJ() { return j1 \* j2 + 5; }

protected int j1 { get; set; }

protected int j2 { get; set; }

}

public class K : J

{

public K() { this.k = 100; }

~K() { }

public new void l() { Console.WriteLine(+10); }

public override void mJ() { this.j1 = this.j1 + this.j2; }

public int k { get; set; }

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Спецификация:");

Console.WriteLine("Объект интерфейса A, созданного с помощью коструктора класса С:");

A a1 = new C();

Console.WriteLine(a1.fA());

a1.mA();

//Console.WriteLine(a1.fC());

//Console.WriteLine(a1.mC());

//Console.WriteLine(a1.S());

//Console.WriteLine(a1.c);

//Console.WriteLine(a1.c1);

Console.WriteLine();

Console.WriteLine("Объект интерфейса A, созданного с помощью коструктора класса С:");

A a2 = new K();

Console.WriteLine(a2.fA());

a2.mA();

//Console.WriteLine(a2.fC());

//Console.WriteLine(a2.mC());

//Console.WriteLine(a2.l());

//Console.WriteLine(a2.mJ());

//Console.WriteLine(a2.pJ());

//Console.WriteLine(a2.k);

//И. Т. Д.

Console.ReadKey();

Console.WriteLine("Расширение:");

//расширение

C c = new C();

Console.WriteLine("Класс C функция c.fA = {0}", c.fA());

Console.WriteLine("Класс C функция b.fC = {0}", c.fC());

Console.WriteLine("Значение атрибута доступа c1 класса B = {0} ", c.c1);

c.mC();

c = new D(); // неполная подстановка объекта класса D в объект c

//расширение по функции

Console.WriteLine("Класс D функция c.fC = {0}", c.fC());

c.mC();

//Console.WriteLine(c.fD());

Console.ReadKey();

//Специализация

Console.WriteLine("Специализация:");

C c1 = new E();

Console.WriteLine(c1.fC());

c1.mC();

Console.WriteLine(c1.fA());

c1.mA();

Console.WriteLine(c1.S());

//Console.WriteLine(c1.c);

//Console.WriteLine(c1.e);

E e1 = new E();

Console.WriteLine(e1.fC());

e1.mC();

Console.WriteLine(e1.fA());

e1.mA();

Console.WriteLine(e1.S());

Console.WriteLine(e1.e1);

//Console.WriteLine(e1.c);

Console.ReadKey();

//конструирование

Console.WriteLine("Конструирование:");

J j = new K();

j.l();

((K)j).l();

K k = new K();

k.l();

((J)k).l();

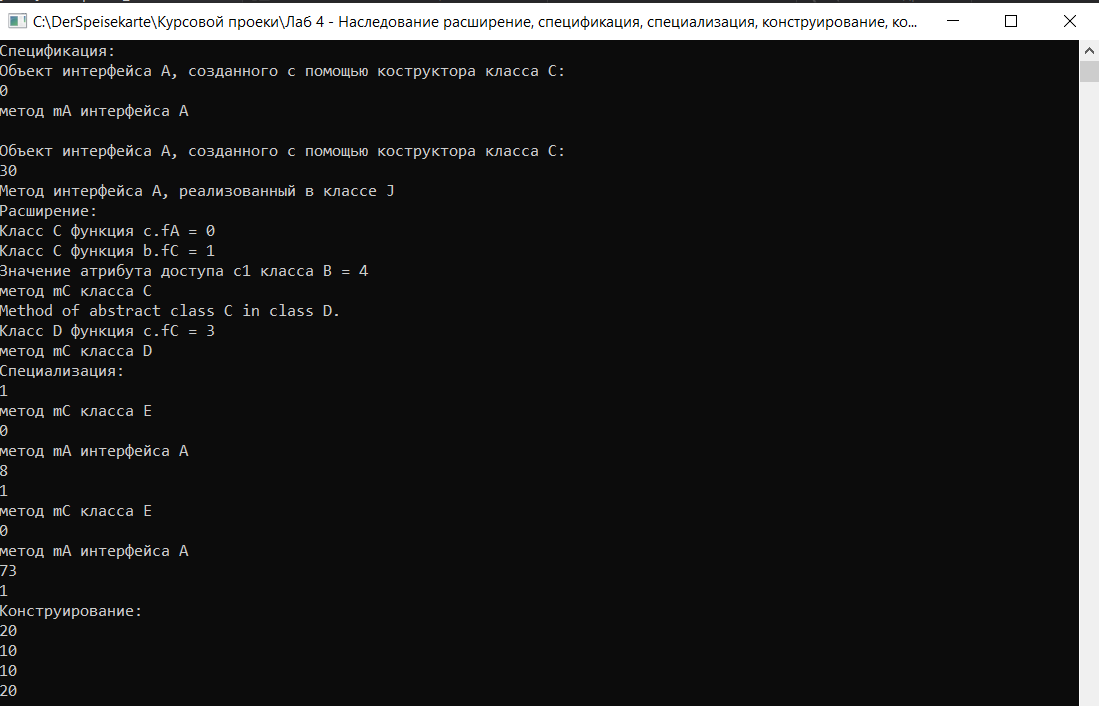
Console.ReadKey();

}

}

}

**Результат:**



**Вывод:**

За счет того, что при расширении не меняется наследуемое поведение, дочерний класс только лишь обогащает родительский класс новыми полями и методами. При это дочерний класс можно рассматривать как объект родительского класса, ибо он реализует все то, что реализует родительский класс. В нашем примере родительским классом был простой калькулятор для работы с матрицами, а дочерним – более продвинутый калькулятор.

**4.2 Комбинирование**

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSharpLab4\_2

{

public interface F

{

void mF();

int fF();

}

public interface E

{

int X();

void mE();

}

public class D

{

public D()

{

this.speed = 33;

}

~D() { }

public int speed { get; set; }

//расширение по функции

public virtual int X()

{

Console.WriteLine();

return 1;

}

}

//комбинирование

public class C : D, E, F

{

public C() { this.size = 22; this.color = 1; }

~C() { }

protected int size { set; get; }

public int color { set; get; }

public void mF() { this.color = this.color \* this.size; }

public int fF() { return this.color = 55; }

public override int X() { return this.size = 22; }

public void mE() { this.color = this.size; Console.WriteLine("method mE of class C"); }

public void mD() { Console.WriteLine("it is very interesting"); }

public void mC() { Console.WriteLine("method mC of class C"); }

}

class Program

{

static void Main(string[] args)

{

F f = null;

f = new C();

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

Console.WriteLine("method mF() of Interface F:");

f.mF();

Console.WriteLine("f.X() = {0}", ((D)f).X());

Console.WriteLine();

C c = new C();

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

Console.WriteLine("method mE() ");

c.mE();

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

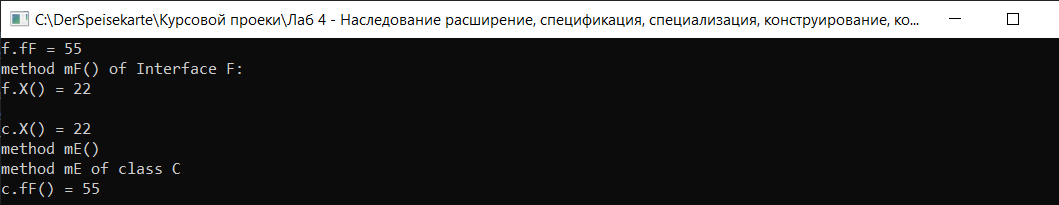
Console.ReadKey();

}

}

}

**Результат:**



**Вывод:**

Комбинирование позволяет объединить черты нескольких классов и интерфейсов в одном дочернем классе. В моей программе реализовано наследование от двух интерфейсов *E* и *F* и класса *K*. Происходит множественное наследование, причем для каждого суперкласса можно определить свой вариант наследования, но, если в наследуемых классах есть одинаковые методы и/или переменные, неоднозначность использования определяется указанием пространства имен класса, переменную или метод которого нужно использовать.