Московский Авиационный Институт

(Национальный Исследовательский Университет)

**Факультет информационных технологий и прикладной математики**

Кафедра вычислительной математики и программирования

Лабораторная работа 4 по курсу ООП:

Основы программирования на языке C#

4.НАСЛЕДОВАНИЕ: РАСШИРЕНИЕ, СПЕЦИФИКАЦИЯ, СПЕЦИАЛИЗАЦИЯ, КОНСТРУИРОВАНИЕ.

Работу выполнил:

М8О-205Б-21 Жилов А.А. \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

Руководитель: \_\_\_\_\_\_\_\_\_\_\_/Кузнецова С.В.

Дата: \_\_\_Октября 2022

**Расширение, спецификация, специализация, конструирование**

**Текст программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection.Metadata.Ecma335;

using System.Runtime.Intrinsics.X86;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace lab\_4\_1

{

public interface A

{

void mA();

int fA();

}

//расширение

public class B : A

{

public B()

{

this.v\_1 = 1;

this.v\_2 = 2;

}

protected int v\_1 { set; get; }

public int v\_2 { set; get; }

public virtual int f()

{

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

return 0;

}

public void mA()

{

this.v\_1 += 1;

}

public int fA()

{

return v\_2 + 2;

}

}

public class J : A

{

public J()

{

this.v\_1 = 1;

}

protected int v\_1 { set; get; }

public virtual int f()

{

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

return 0;

}

public void mA()

{

this.v\_1 += 1;

}

public int fA()

{

return v\_1 + 2;

}

}

//специализация

public class D : B

{

public D()

{

this.v\_3 = 3;

}

public int v\_3 {set; get;}

public override int f()

{

Console.WriteLine("Class D f()");

return base.f()+7;

}

public int fD()

{

return this.v\_1 + this.v\_3;

}

}

//спецификация

public abstract class C :A

{

public void mC(int a)

{

this.v\_1 = a;

}

private int v\_1 = 11;

public abstract int fC();

public void mA() { this.v\_1 += this.v\_1 \* 2; }

public int fA() { return this.v\_1; }

}

public class F:C

{

public F() { }

public override int fC()

{

Console.WriteLine("fC()");

return 0;

}

}

public class E :C

{

public E() { }

public override int fC()

{

Console.WriteLine("fC(");

return 5 + this.fA();

}

}

class Program

{

static void Main(string[] args)

{

A a = new B();

B b = new B();

Console.WriteLine("Extension");

Console.WriteLine($"class B b.f() {b.f()}");

Console.WriteLine($"class B v\_2 {b.v\_2}");

a.mA();

Console.WriteLine($"interface A fA() {a.fA()}");

a = new J();

a=b;

Console.WriteLine($"class J b.f() {b.f()}");

Console.WriteLine($"class J v\_2 {b.v\_2}");

a.mA();

Console.WriteLine($"interface A fA() {a.fA()}");

Console.WriteLine("Specialization");

b =new D();

a = b;

Console.WriteLine($"class D b.f() {b.f()}");

Console.WriteLine($"class D v\_2 {b.v\_2}");

Console.WriteLine($"class D v\_3 {((D)b).v\_3}");

Console.WriteLine($"class D fD() {((D)b).fD()}");

Console.WriteLine("Specification");

C c = null;

c = new E();

Console.WriteLine("class C c.fC() {0}", c.fC());

c = new F();

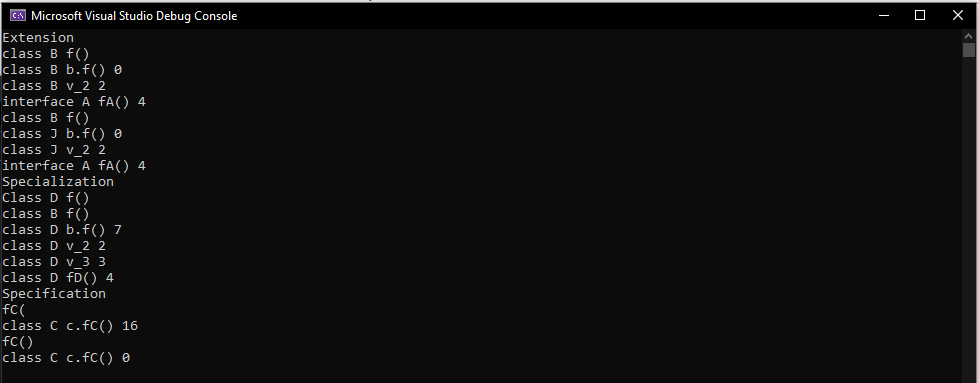
Console.WriteLine("class C c.fC() {0}", c.fC());

}

}

}

**Результат работы**

****

**Комбинирование**

**Текст программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab\_4\_1

{

public interface B

{

void mB();

int fB();

}

public class C

{

public C()

{

this.v\_1 = 12;

}

public int v\_1 { get; set; }

public int f()

{

Console.WriteLine("class C f()");

return 2;

}

}

public class E:C,B

{

public E() { this.v\_2 = 5; this.v\_3 = 10; }

protected int v\_2 { set; get; }

public int v\_3 { set; get; }

public int fB() { return this.v\_3 \* (10 - this.v\_1); }

public void mB() { this.v\_2 = this.v\_1 \* this.v\_3 + 100; }

}

class Program

{

static void Main(string[] args)

{

E a = new E();

Console.WriteLine("combination");

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

Console.WriteLine($"((A)b).f() = {((E)a).f()}");

C c = new C();

c.f();

c = new E();

Console.ReadKey();

}

}

}

**Результат работы**



**Вывод**

При специализации дочерний класс является частным случаем родительского класса. Расширение дает возможность добавить новые функциональные возможности к родительскому классу, но не меняет наследуемое поведение. При спецификации родительский класс описывает поведение, которое реализуется в дочернем классе, но оставлено нереализованным в родительском. Во время конструирования дочерний класс использует методы, предопределяемые родительским классом. Комбинирование позволяет объединить черты нескольких классов или интерфейсов в одном.