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, КОМБИНИРОВАНИЕ

Работу выполнил:

М8О-205Б-21 Титеев Р.М. _____ _____

(*подпись*) (*вариант*)

Руководитель: _____/Кузнецова С.В.

(*подпись*)

Дата: ____ октября 2022

Расширение, Спецификация, Специализация, Конструирование

Текст программы

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

namespace lab_4_1
{
    public interface A  {
        void mA();
        int fA();
    }

    // Specialization + Extension
    public class B : A {
        public B() {
            this.v_1 = 10;
            this.v_2 = 20;
        }
        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 = this.v_1 * this.v_1;
        }
        public int fA() {
            return this.v_2 + 1;
        }
    }
    public class D : B {
        public D(){
            this.v_3 = 33;
        }
        public int v_3 {set; get;}
        public override int f() {
            Console.WriteLine("class D f() ");
            return base.f() + 10;
        }
        public int fD() {
            return this.v_1 * this.v_3;
        }
```

```
45      }
46
47      // Specification
48      public abstract class C : A  {
49          public void mC(int a){
50              this.v_1 = a;
51          }
52          public abstract int fC();
53          public void mA() { this.v_1 = this.v_1 * 10; }
54          public int fA() { return this.v_1; }
55          private int v_1 = 22;
56      }
57      public class E : C  {
58          public E() { }
59          public override int fC(){
60              Console.WriteLine("fC()");
61              return 0;
62          }
63      }
64      public class J : C {
65          public J() { }
66          public override int fC() {
67              Console.WriteLine("fC()");
68              return 1 + this.fA();
69          }
70      }
71
72      // Construction
73      public class F{
74          public F() {
75              this.v_1 = 1;
76          }
77          protected int v_1 { set; get;}
78          public int f() {
79              return v_1;
80          }
81      }
82      public class K : F {
83          public K() {
84              this.v_2 = 2;
85          }
86          private int v_2 { set; get;}
87          public int f() {
88              return v_2 + v_1;
89          }
90      }
91
92
```

```
93      internal class Program
94      {
95          static void Main(string[] args)
96          {
97
98
99              A a = new B();
100             B b = new B();
101             Console.WriteLine("    Specialization + Extension");
102             Console.WriteLine($"class B b.f() {b.f()}");
103             Console.WriteLine($"class B v_2 {b.v_2}");
104             a.mA();
105             Console.WriteLine($"intrface A fA() {a.fA()}");
106
107             b = new D();
108             a = b;
109
110             Console.WriteLine($"class D b.f() {b.f()}");
111             Console.WriteLine($"class D v_2 {b.v_2}");
112             Console.WriteLine($"class D v_3 {((D)b).v_3}");
113             Console.WriteLine($"class D fD() {((D)b).fD()}");
114
115
116             Console.WriteLine("    Specification");
117             C c = null;
118             c = new E();
119             Console.WriteLine("class C c.fC() {0}", c.fC());
120
121             c = new J();
122             Console.WriteLine("class C c.fC() {0}", c.fC());
123
124
125             Console.WriteLine("    Construction");
126             F f = new F();
127             Console.WriteLine($"class F f.f() {f.f()}");
128             f = new K();
129             Console.WriteLine($"class K f.f() {((K)f).f()}");
130
131             Console.ReadKey();
132         }
133     }
134 }
```

Результат работы

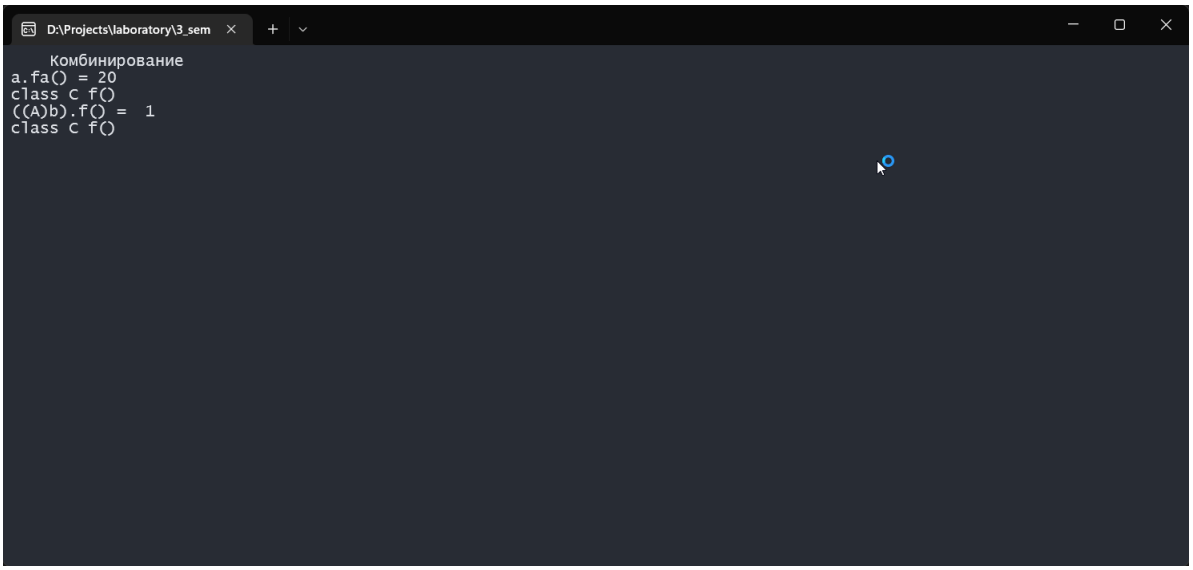
Комбинирование

Текст программы

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

namespace lab_4_2
{
    public interface A{
        void mA();
        int fA();
    }

    public interface B{
        int fB();
        void mB();
    }

    public class C{
        public C() { this.v_1 = 33; }
        public int v_1 { set; get; }
        public int f()
        {
            Console.WriteLine("class C f() ");
            return 1;
        }
    }

    public class D : C, A, B{
        public D() { this.v_2 = 1; this.v_3 = 2; }
        protected int v_2 { set; get; }
        public int v_3 { set; get; }

        public void mA() { this.v_2 = this.v_3 + this.v_1; }
        public int fA() { return this.v_3 * 10; }

        public int fB() { return this.v_3 * (10-this.v_1); }
        public void mB() { this.v_2 = this.v_1*this.v_3+100; }
    }
    internal class Program{
        static void Main(string[] args)
        {
            A a = null;
            a = new D();
```

```
45              a.mA();
46              Console.WriteLine("    Combination");
47              Console.WriteLine($"a.fa() = {a.fA()}");
48              Console.WriteLine($"((A)b).f() =  {((D)a).f()}");
49
50              C c = new C();
51              c.f();
52              c = new D();
53
54              Console.ReadKey();
55
56          }
57      }
58  }
```

Результат работы



Вывод

При специализации дочерний класс является более конкретным, частным или специализированным 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 конструирование дочерний класс использует методы, предопределяемые родительским классом.

Комбинирование позволяет объединить черты нескольких классов(интерфейсов) в одном (в данном случае трех).