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

**Диаграмма классов**

Protected

Private

Public

F

Protected

Private

Public

E

Protected

Private

Public

K

Protected

Private

Public

C

Protected

Private

Public

B

Protected

Private

Public

J

Protected

Private

Public

A

Спецификация Расширение

B

A

**Protected**

**Private**

**Public**

**Protected**

**Private**

**Public**

K

**Public**

Void mA()

a

b

Int fA() int fB()

**Private**

**Protected**

B

b

**Public**

Void mA() int fB()

a

Int fA()

b

**Private**

Void mA()

b

Int fK() int fB()

Int fA()

**Protected**

Конструирование

Специализация

**Protected**

**Private**

**Public**

**Protected**

**Private**

**Public**

**Protected**

**Private**

**Public**

**Protected**

**Private**

**Public**

F

C

С

Int fC()

Void l()

c

c Void l()

С

Int fC()

f Void l()

F

Int fC()

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_1

{ //спецификация

public interface A

{

void mA();

int fA();

}// классы в которых функции определяются но не реализуются

// В интерфейсах нет реализации и нет полей, не прописаны алгоритмы функций и не параметров классов и поэтому нельзя создать объекты.

public class B : A

{

public B() { this.b = 1; this.b1 = 4; }

~B() { }

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

public int fA()

{

this.b += 1;

Console.WriteLine();

return b;

}

public virtual int fB() { return 1; }

public virtual void mB() { Console.WriteLine("метод mB класса B"); }

protected int b { get; set; }

public int b1 { get; set; }

}//конец спецификации

// Наследование интерфейса и абстрактных функций

//расширение

public class K : B

{

public K()

{

this.k1 = 1;

this.k2 = 2;

}

public override int fB()// замещение

{

Console.WriteLine("Method of abstract class B in class K.");

return this.k1 \* this.k2 + base.fB();

}

~K() { }

public int fK()// новый метод

{

Console.WriteLine("Method of K");

return this.k1 + this.k2;

}

public override void mB() { Console.WriteLine("метод mB класса K"); }

public int k1 { get; set; }

public int k2 { get; set; }

}//конец расширения

public abstract class C : A //абстрактный класс и у него есть конструктор но не можем создать объекты. Например: нельзя сделать так: C c = new C();

{

public C()

{

this.c1 = 10;

this.c2 = 20;

}

~C() { }

public void l() { Console.WriteLine(+50); } // операция, но лучше говорить метод ( void )

public void mA() { }

public int fA() { return c1 + c2; }

public abstract void mC();

public abstract int fC();

public virtual int pC() { return c1 \* c2 + 5; }

protected int c1 { get; set; }

protected int c2 { get; set; }

}

public class E : C

{

public E() { this.e = 100; }

~E() { }

public override void mC() { this.c1 = this.c1 + this.c2; }

public override int fC()

{

Console.WriteLine("Function of E");

return c1 + e;

}

public override int pC() { Console.WriteLine("function pC classa E"); return c1 \* c2 + 10; } //Переопределения фунцкии базового класса в подклассе ( методы) или замещена

public int e { get; set; }

}

public class F : C

{

public F() { this.f = 100; }

~F() { }

public new void l() { Console.WriteLine(+25); }// с помощью new мы скрываем метод базового класса

public override void mC() { this.c1 = this.c1 + this.c2; }

public override int fC()

{

Console.WriteLine("Function of E");

return c1 + f;

}

protected virtual int pC() { Console.WriteLine("function pC classa F"); return 100; }

public int f { get; set; }

}

public class J : A

{

public J()

{

this.j1 = 1;

this.j2 = 2;

this.j3 = 3;

}

~J() { }

public void mA() { }

public int fA() { return j1 \* j2 \* j3; }

protected int j1 { get; set; }

protected int j2 { get; set; }

protected int j3 { get; set; }

}

class Program

{

static void Main(string[] args)

{

A a1 = new B();//создаём объект интерфейса A с помощью конструктора класса B

A a2 = new J();

A a3 = new K();

//спецификация

Console.WriteLine("Спецификация:");

Console.WriteLine("interface A a2.fA() = {0}", a2.fA());

//Создаём объекты класса C с помощью конструкторов класса E и F

Console.WriteLine();

Console.WriteLine("Специализация:");//

C c1 = new E();

E e1 = new E();

c1.fC();

e1.fC();

Console.WriteLine();

//подстановка

Console.WriteLine("Расширение:");

//Создаём объект класса B

B b = new B();

Console.WriteLine("класс B функция b.fA = {0}", b.fA());

Console.WriteLine("класс B функция b.fB = {0}", b.fB());

Console.WriteLine("значение атрибута доступа b1 класса B = {0} ", b.b1);

b.mB();

b = new K();

//расширение по функции

// добавление новых методов в подкласс, а расширение по фунцкии - добавление новых алгоритмов

Console.WriteLine("класс K функция b.fB = {0}", b.fB());

b.mB();

Console.WriteLine();

//конструирование

// Это в другом классе создание одноимённой функции и она не является похожей на предыдущую

// Это сокрытие фукнции базового класса и заменой её новой функцией в подклассе(наличие тех же параметров)

Console.WriteLine("Конструирование:");

C c = new F();

c.l();

F f = new F();

f.l();

Console.WriteLine();

C e = new E();

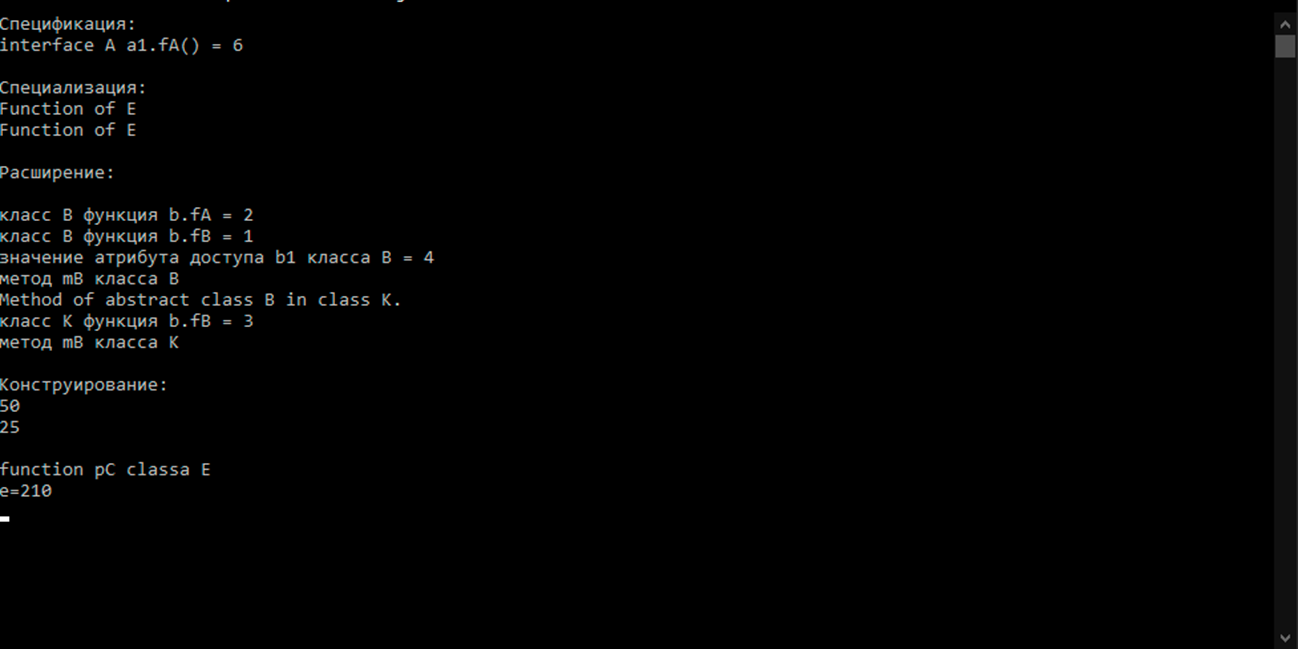
Console.WriteLine("e={0}", e.pC());

Console.ReadKey();

}

}

}



**Расширение**. В подобъект добавляются совершенно новые свойства, замещается, по крайней мере, один метод объекта суперкласса. Добавляются новые методы к объекту подкласса. Функциональные возможности менее крепко связаны с объектом суперкласса. Принцип подстановки выполняется не полностью, потому что в подобъект добавляются новые свойства и методы.

**Специализация.** Объект подкласса является более конкретным, частным или специализированным случаем для объекта суперкласса. Объект подкласса удовлетворяет спецификации объекта суперкласса. Принцип подстановки выполняется полностью.

**Спецификация.** Спецификация гарантирует поддержку объектами определенного общего интерфейса. Объект суперкласса A описывает поведение, которое должно быть реализовано в объекте подкласса. Принцип подстановки выполняется между подклассами, сохраняющими данный интерфейс суперкласса {B,C, ...}.

**Конструирование.** В подклассе изменяются имена методов, предоставляемые суперклассом, или модифицируются аргументы, объекты подкласса не являются подтипом суперкласса. Принцип подстановки не выполняется.

**Комбинирование.**

A

Public

Private

Protected

Protected

Private

Public

J

Protected

Private

Public

B

Protected

Private

Public

C

**c**

Int fC()

Void mC()

**b**

Int K()

Void mB()

**j**

Int K()

Int K(); void mB();

b

a

c

j

Int K() Int fC(); void mC();

Void mA()

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_2

{

public interface C

{

void mA();

int fA();

}

public interface B

{

int K();

void mB();

}

public class J

{

public J()

{

this.speed = 33;

}

~J() { }

public int speed { get; set; }

//расширение по функции

public virtual int K()

{

Console.WriteLine();

return 1;

}

}

//комбинирование

public class A : J, C, B

{

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

~A() { }

protected int size { set; get; }

public int color { set; get; }

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

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

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

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

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

}

class Program

{

static void Main(string[] args)

{

C c = null;// пустая ссылка

c = new A();

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

Console.WriteLine("method mA() of Interface A:");

c.mA();

Console.WriteLine("a.K() = {0}", ((J)c).K());// всё тоже самое как J c = new J() и вызов потом c.K() или по другому это преобразование типов

Console.WriteLine();

A d = new A();

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

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

d.mB();

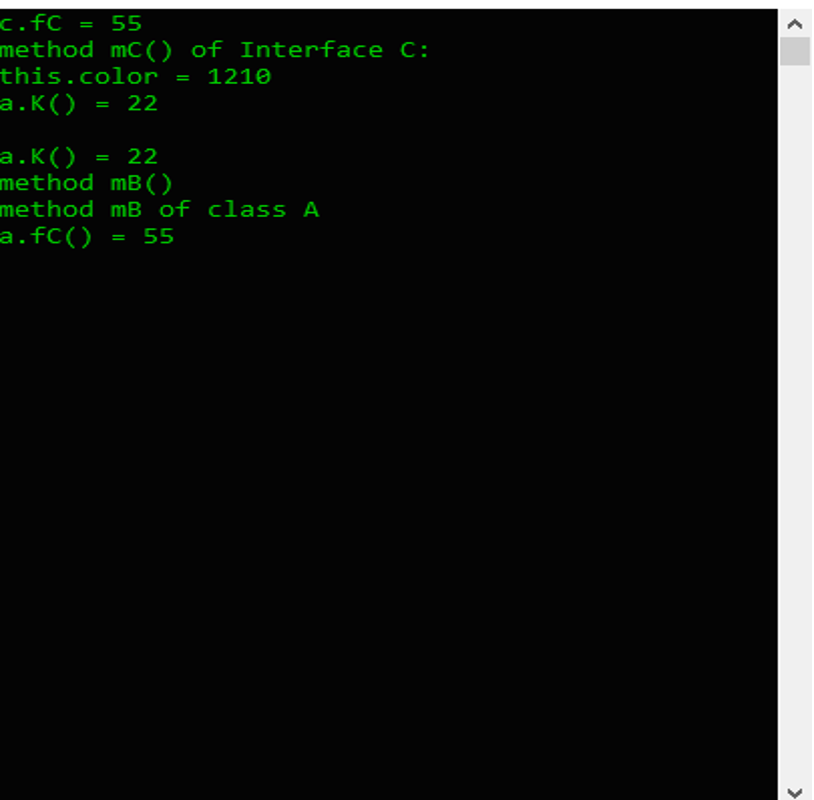
Console.WriteLine("d.fA() = {0}", d.fA());

Console.ReadKey();

}

}

}



Вывод: Комбинирование позволяет объединить черты нескольких классов и интерфейсов в одном дочернем классе. 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.