**Лабораторная работа №7. Использование. Клиент-сервер.**

**Диаграмма классов:**

Protectededd

Private

Public

E

Protectededd

Private

Public

D

Клиент Сервер

 Use

Текст программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab7\_cs

{

class Program

{

class D//клиент

{

public D() { Console.WriteLine("Constructor D"); }

~D() { }

public int f()

{

this.d = 5;

return this.d \* 2;

}

public int F()

{

this.d = 15;

return this.d + 5;

}

public int d { set; get; }

}

class E//сервер

{

public E() { Console.WriteLine("Constructor E"); }

~E() { }

public void me(D d) { Console.WriteLine("class E client me() = {0}", d.f()); }

public void ME(D d) { Console.WriteLine("Class E client ME() = {0}", d.F()); }

}

static void Main(string[] args)

{

D d = new D();

E e = new E();

e.me(d); // клиент(объект d) обращается к ресурсам сервера(класс E) через объект e с помощью метода me.

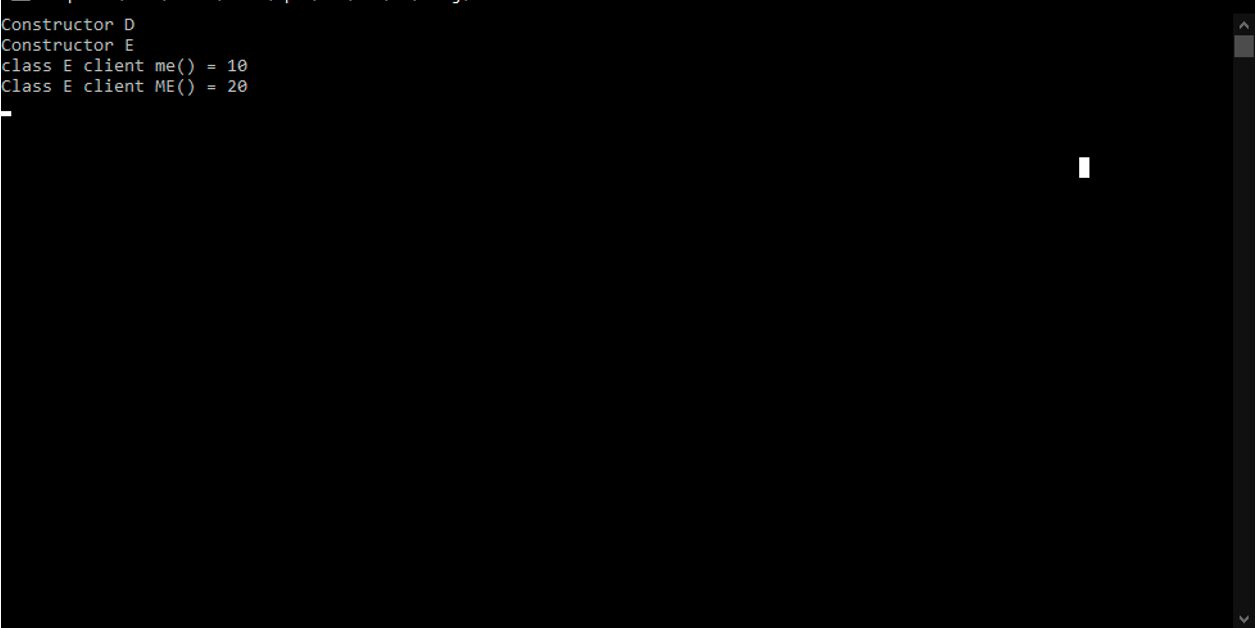
e.ME(d);

Console.ReadKey();

}

}

}

Вывод: С помощью использования клиент-сервер, можно контролировать или управлять одним объектом – клиентом, через другой объект – сервер. При это объект-клиент выступает в качестве параметра для объекта-сервера. Клиент и сервер могут находиться как на одном компьютере(ПК), так и на разных ПК в сети. Также может возникать такая ситуация, когда некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому.

**Общность.**

private

Public

E

public

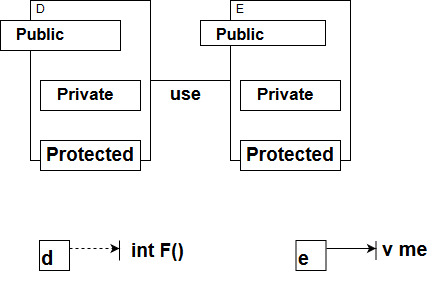
private

D

use

protected

protecteddв

****

Текст программы :

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab7\_friend

{

class Program

{

static class D

{

//public static D() { }// не может содержать конструкторы

public static int F() { return 7; } // Содержит только статические член

//public void f() { Console.WriteLine("Нельзя это использовать в статическом классе..."); }

}// Статический класс в основном такой же, как и нестатический класс, но имеется одно отличие: нельзя создавать экземпляры статического класса.

class F

{

public static int R() { return 10; } // а так можно в обычном классе использовать статические функции

private static int F1() { return 1; }

}// статические функции можно использовать в других классах и функциях и они также будут доступны везде

// скрыть их можно с помощью модификатора доступа private

class E

{

public E() { }

~E() { }

public void me() { Console.WriteLine("class D function F = {0}", D.F()); }

public void ME() { Console.WriteLine("class F function R = {0}", F.R()); }

}

static void Main(string[] args)

{

F f = new F();

E e = new E();

e.me();

//нельзя сделать D d = new D();

D.F();

Console.WriteLine("class F function R = {0}", F.R());

e.ME();

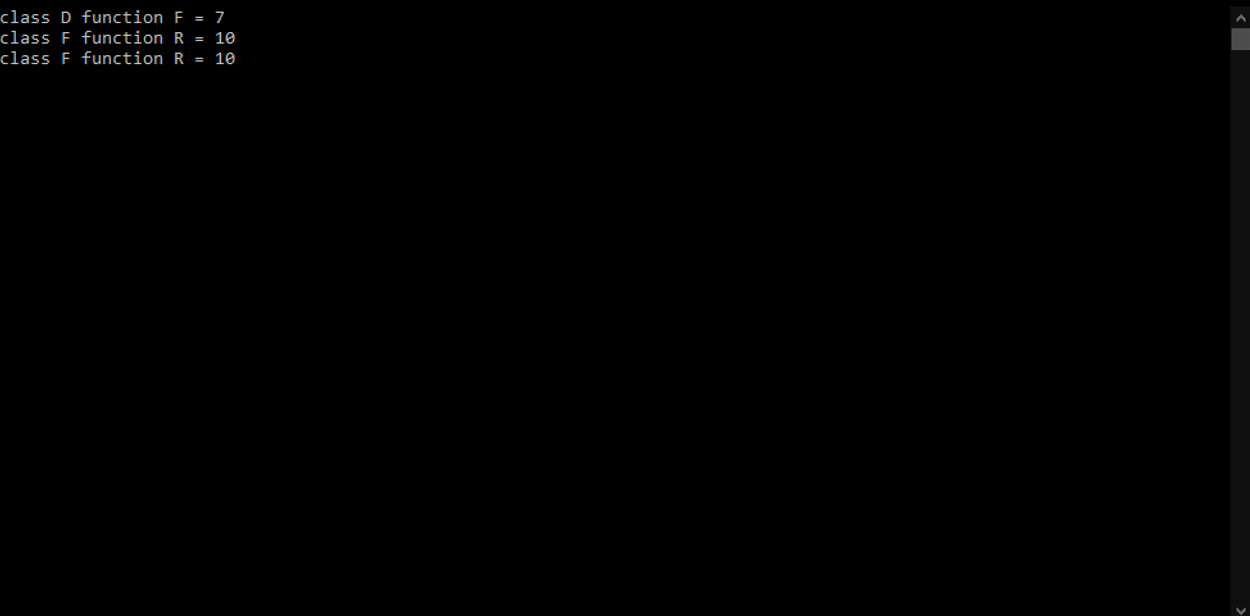
//Console.Readkey() — Console - класс. а ReadKey() - статическая функция

Console.ReadKey();

}

}

}



**Вывод:** Использование общности позволяет одному объекту взаимодействовать со всеми методами и членами второго, если первый объект по отношению ко второму объявляется другом. Также, чтобы открыть доступ функциям к данным объекта. они могут быть объявлены дружественными.