Московский авиационный институт

(Национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

Кафедра вычислительной математики и программирования

**Лабораторные работы 1 — 9 по курсу ООП:  
основы программирования на С#**

0.Перегрузка

1.Агрегация по ссылке

2.Агрегация по значению и вложением

3.Принцип подстановки

4.Наследование: расширение, спецификация, специализация, конструирование и комбинирование

5.Наследование: комбинирование через общих предков

6.Ассоциация(один к одному, один ко многим)

7.Использование

8.Конкретизация

9. АНОНИМНЫЕ ФУНКЦИИ: delegate, ЛЯМБДА ВЫРАЖЕНИЕ. Event

Работу выполнил:

8О-203Б ........................ \_\_\_\_\_\_\_\_\_ Вариант 9

Руководитель: \_\_\_\_\_\_\_/Семенов А.С.

Дата: \_\_\_ октября 2019

**Перегрузка конструкторов, функций, операторов и операций**

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab0\_overloading {

class Program {

class custom {

public int atrib\_1, atrib\_2;

public custom() {

Console.WriteLine("constructor 1");

}

public custom(int atrib\_1) {

Console.WriteLine("constructor 2");

this.atrib\_1 = atrib\_1;

this.atrib\_2 = 0;

}

public custom(int atrib\_1, int atrib\_2) {

Console.WriteLine("constructor 3");

this.atrib\_1 = atrib\_1;

this.atrib\_2 = atrib\_2;

}

~custom() {}

public int func(int x) {

return x;

}

public int func(int x, int y) {

return x + y;

}

public int func(int x, float y) {

return x + (int)y;

}

public static custom operator +(custom c\_1, custom c\_2) {

custom c\_res = new custom(

c\_1.atrib\_1 + c\_2.atrib\_1,

c\_1.atrib\_2 + c\_2.atrib\_2);

return c\_res;

}

public static custom operator -(custom c\_1, custom c\_2) {

custom c\_res = new custom(

c\_1.atrib\_1 - c\_2.atrib\_1,

c\_1.atrib\_2 - c\_2.atrib\_2);

return c\_res;

}

public static bool operator >(custom c\_1, custom c\_2) {

return c\_1.atrib\_1 > c\_2.atrib\_2;

}

public static bool operator <(custom c\_1, custom c\_2) {

return c\_1.atrib\_1 < c\_2.atrib\_2;

}

public void oper(int x, int y) {

this.atrib\_1 = x + y;

}

public void oper(int x) {

this.atrib\_1 = x;

}

}

static void Main(string[] args) {

custom c\_1 = new custom();

custom c\_2 = new custom(5);

custom c\_3 = new custom(3, 6);

Console.WriteLine(

"c\_1 is empty, c\_2 atrib\_1 = {0}, c\_2 atrib\_2 = {1}, c\_3 atrib\_1 = {2}, c\_3 atrib\_2 = {3}",

c\_2.atrib\_1, c\_2.atrib\_2, c\_3.atrib\_1, c\_3.atrib\_2

);

Console.WriteLine(

"c\_2 func 1 = {0}, 2 = {1}, 3 = {2}", c\_2.func(2),

c\_2.func(1, 2), c\_2.func(1, (int)2.3)

);

Console.WriteLine(

"c\_3 func 1 = {0}, 2 = {1}, 3 = {2}", c\_3.func(2),

c\_3.func(1, 2), c\_3.func(1, (int)2.3)

);

custom c\_sum = new custom();

c\_sum = c\_2 + c\_3;

Console.WriteLine("c\_sum atrib\_1 = {0}, atrib\_2 = {1}", c\_sum.atrib\_1, c\_sum.atrib\_2);

c\_sum = c\_2 - c\_3;

Console.WriteLine("c\_diff atrib\_1 = {0}, atrib\_2 = {1}", c\_sum.atrib\_1, c\_sum.atrib\_2);

bool c\_comp = c\_sum > c\_3;

Console.WriteLine(c\_comp);

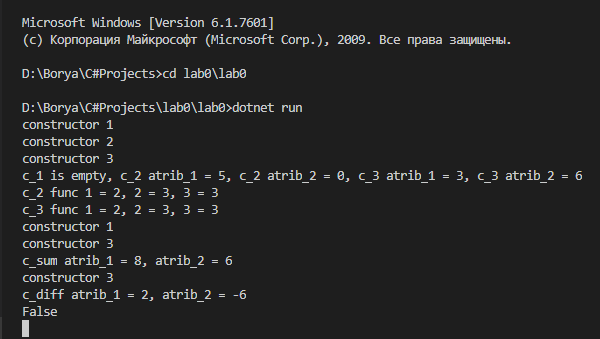
Console.ReadKey();

}

}

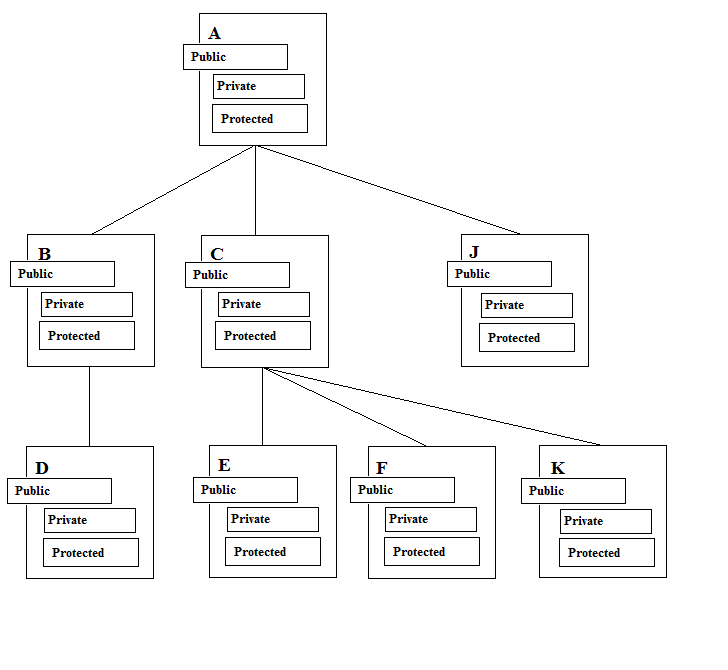
}

**Результат работы:**

****

**Вывод:** перегрузка конструкторов, функций, операторов и операций предоставляет возможности определения нескольких функций с одинаковым именем, разных конструкторов для тех или иных классов, а так же элементарных арифметических операций с этими классами.

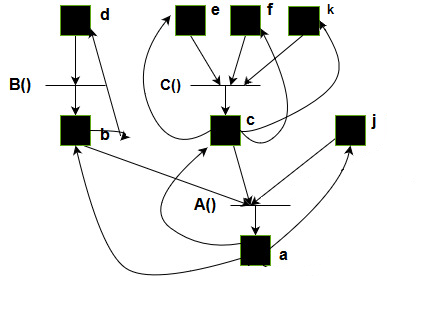
**Лабораторная работа №1. Агрегация по ссылке**

****

**Рис.1.** диаграмма классов: агрегация по ссылке.

**Текст программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab1

{

class Program

{

class custom\_a {

private custom\_b atrib\_b = null;

private custom\_c atrib\_c = null;

private custom\_j atrib\_j = null;

public custom\_a(custom\_b atrib\_b, custom\_c atrib\_c, custom\_j atrib\_j) {

this.atrib\_b = atrib\_b;

this.atrib\_c = atrib\_c;

this.atrib\_j = atrib\_j;

atrib\_c.cq = 24;

}

~custom\_a() {}

public void a\_method() {

Console.WriteLine("method of a");

}

public custom\_b set\_get\_atrib\_b {

set {

Console.WriteLine("set b");

atrib\_b = value;

}

get {

Console.WriteLine("get b: ");

return atrib\_b;

}

}

public custom\_c set\_get\_atrib\_c {

set {

Console.WriteLine("set c");

atrib\_c = value;

}

get {

Console.WriteLine("get c: ");

return atrib\_c;

}

}

public custom\_j set\_get\_atrib\_j {

set {

Console.WriteLine("set j");

atrib\_j = value;

}

get {

Console.WriteLine("get j: ");

return atrib\_j;

}

}

}

class custom\_b {

private custom\_d atrib\_d = null;

public custom\_b(custom\_d atrib\_d) {

this.atrib\_d = atrib\_d;

}

~custom\_b() {}

public void b\_method() {

Console.WriteLine("method of b");

}

public custom\_d set\_get\_atrib\_d {

set {

Console.WriteLine("set d");

atrib\_d = value;

}

get {

Console.WriteLine("get d: ");

return atrib\_d;

}

}

}

class custom\_c {

private custom\_e atrib\_e = null;

private custom\_f atrib\_f = null;

private custom\_k atrib\_k = null;

public custom\_c(custom\_e atrib\_e, custom\_f atrib\_f, custom\_k atrib\_k) {

this.atrib\_e = atrib\_e;

this.atrib\_f = atrib\_f;

this.atrib\_k = atrib\_k;

}

~custom\_c() {}

public void c\_method() {

Console.WriteLine("method of c");

}

public custom\_e set\_get\_atrib\_e {

set {

Console.WriteLine("set e");

atrib\_e = value;

}

get {

Console.WriteLine("get e: ");

return atrib\_e;

}

}

public custom\_f set\_get\_atrib\_f {

set {

Console.WriteLine("set f");

atrib\_f = value;

}

get {

Console.WriteLine("get f: ");

return atrib\_f;

}

}

public custom\_k set\_get\_atrib\_k {

set {

Console.WriteLine("set k");

atrib\_k = value;

}

get {

Console.WriteLine("get k: ");

return atrib\_k;

}

}

public int cq {get; set;}

}

class custom\_d {

public custom\_d() {}

~custom\_d() {}

public void d\_method() {

Console.WriteLine("method of d");

}

}

class custom\_e {

public custom\_e() {}

~custom\_e() {}

public void e\_method() {

Console.WriteLine("method of e");

}

}

class custom\_f {

public custom\_f() {}

~custom\_f() {}

public void f\_method() {

Console.WriteLine("method of f");

}

}

class custom\_j {

public custom\_j() {}

~custom\_j() {}

public void j\_method() {

Console.WriteLine("method of j");

}

}

class custom\_k {

public custom\_k() {}

~custom\_k() {}

public void k\_method() {

Console.WriteLine("method of k");

}

}

static void Main(string[] args)

{

custom\_d c\_d = new custom\_d();

custom\_e c\_e = new custom\_e();

custom\_f c\_f = new custom\_f();

custom\_k c\_k = new custom\_k();

custom\_j c\_j = new custom\_j();

custom\_b c\_b = new custom\_b(c\_d);

custom\_c c\_c = new custom\_c(c\_e, c\_f, c\_k);

custom\_a c\_a = new custom\_a(c\_b, c\_c, c\_j);

Console.WriteLine("access atribute: {0}", c\_c.cq);

Console.WriteLine("link aggregation");

c\_a.set\_get\_atrib\_b.b\_method();

c\_a.set\_get\_atrib\_c.c\_method();

c\_a.set\_get\_atrib\_j.j\_method();

c\_a.set\_get\_atrib\_b.set\_get\_atrib\_d.d\_method();

c\_a.set\_get\_atrib\_c.set\_get\_atrib\_e.e\_method();

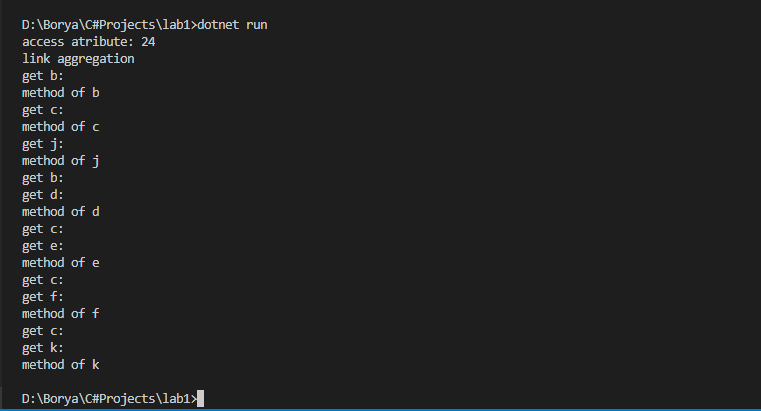
c\_a.set\_get\_atrib\_c.set\_get\_atrib\_f.f\_method();

c\_a.set\_get\_atrib\_c.set\_get\_atrib\_k.k\_method();

}

}

}

**Результат работы программы:  
**

**Вывод:** В данном случае объекты всех классов существуют независимо друг от друга. При этом многие из объектов являются атрибутами для других объектов. Таким образом, если удалить, например, объект c\_b, объекты c\_d и c\_a останутся в памяти, но у c\_a не будет одного атрибута. Агрегация по ссылке очень выгодна, так как она расходует очень мало памяти.