**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет «Информационных систем и технологий»

Кафедра «Интеллектуальных систем автоматизации и управления»

|  |  |
| --- | --- |
| Направление подготовки: | 09.03.02 - Информационные системы и технологии |
| Направленность (профиль): | Системное и прикладное программирование информационных систем |

**ЛАБОРАТОРНАЯ РАБОТА № 1**

по дисциплине:

|  |
| --- |
| **Объектно-ориентированное программирование на C#** |

на тему:

|  |
| --- |
| **«АССОЦИАЦИЯ, АГРЕГАЦИЯ И КОМПОЗИЦИЯ»** |

Вариант № 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Выполнил студент группы | | ИСТ-261 |
|  |  | Филатов М.Ю. | | |
|  |  | *Фамилия И. О.* | | |
|  |  | Руководитель | … | |
| *оценка* |  |  | *уч. степень, уч. звание* | |
|  |  | Шабанов А.П. | | |
| *дата, подпись* |  | *Фамилия И. О.* | | |

**Цель работы**

Приобретение навыков реализации отношений между объектами в виде ассоциаций, агрегаций и композиций.

**Задачи**

1. Ассоциация, агрегация, композиция кратности «один к одному»
2. Создать два класса A и B, имеющие свойства PropertyA и propertyB соответственно. Реализовать однонаправленную ассоциацию от класса A к классу B кратности «один к одному». Обратиться из объекта класса A к свойству PropertyB объекта класса B.
3. Реализовать между классами двунаправленную ассоциацию кратности «один к одному» путём добавления обратной ссылки в класс B. Создать экземпляры классов A и B. Обратиться из объекта класса A к свойству PropertyB объекта класса B.
4. Реализовать отношение агрегации между классами A и B. Создать экземпляры классов. Реализовать между ними отношение агрегации: добавить ссылку на объект класса B в объект класса A (корень агрегата) c помощью метода Add(B b). Обратиться из объекта класса A к свойству PropertyB объекта класса B.
5. Реализовать отношение композиции между классами A и B. Создать экземпляр класса A. Обратиться из объекта класса A к свойству PropertyB объекта класса B.
6. Однонаправленная ассоциация «один ко многим»

Создать классы «Структурное подразделение» и «Сотрудник» и реализовать между ними однонаправленную ассоциацию от основного класса к зависимому классу кратности «один ко многим». Каждый класс должен содержать не менее одного свойства. Для всех вариантов необходимо переопределить метод ToString(), который будет выводить описание состояния объекта в виде строки. Создать один экземпляр основного класса и пять объектов, являющихся экземплярами зависимого класса. Вызвать метод основного класса ToString(), вывести его результаты в консоль.

1. Двунаправленная ассоциация «один ко многим»

В зависимый класс из эксперимента №2 добавить обратную ссылку. Создать один экземпляр основного класса и три экземпляра зависимого класса, реализовав обратную ссылку в виде свойства навигации. Вызвать метод основного класса ToString(), вывести его результаты в консоль и проанализировать результат его работы. Обратиться из объекта зависимого класса к основному с помощью обратной ссылки и вызвать метод ToString().

1. Отношение агрегации «один ко многим»

Для классов, созданных в результате эксперимента №2, реализовать отношение агрегации. Создать один экземпляр основного класса (корня агрегата) и три экземпляра зависимого класса. Последовательно добавить с помощью метода Add() корня агрегата ссылки на объекты зависимого класса. Вызвать метод основного класса ToString(), вывести его результаты в консоль.

1. Отношение композиции «один ко многим»

Для классов, созданных в результате эксперимента №2, реализовать отношение композиции. Создать один экземпляр основного класса и три экземпляра зависимого класса путем вызова метода New(), в который передаются в качестве параметров значения свойств объектов зависимого 3 класса (объекты зависимых классов создаются внутри метода New()).Вызвать метод основного класса ToString(), вывести его результаты в консоль и проанализировать результат его работы.

1. Глубокое клонирование объектов

Для классов, созданных в результате эксперимента №4, реализовать метод глубокого клонирования, в котором создается клон (копия) объекта, на котором вызывается данный метод, при этом создаются копии всех агрегатов (экземпляров зависимого класса). Создать один экземпляр основного класса (корня агрегата) и три экземпляра зависимого класса. Последовательно добавить с помощью метода Add() корня агрегата ссылки на объекты зависимого класса. Создать клон агрегата, путем вызова метода клонирования на корне агрегата. Вызвать метод ToString() на корне агрегата, вывести его результаты в консоль и проанализировать результат его работы.

**Ход выполнения лабораторной работы**

Для создания UML-диаграмм было использовано веб-приложение draw.io.

**Эксперимент 1**

Созданы классы A и В, имеющие свойства PropertyA и PropertyB соответственно. Для установления отношения ассоциации кратности «один к одному» в классе A, в качестве свойства, указан объект класса B – ObjectB. В файле Program.cs создаются два экземпляра классов A и B, а затем происходит обращение из объекта класса A к свойству PropertyB класса B. Исходный код классов и метода Main представлен на следующих страницах (стр. 4-5).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_1

{

public class A

{

// Properties

public String PropertyA { get; set; }

public B? ObjectB { get; set; }

// Constructor

public A(String propertyA)

{

PropertyA = propertyA;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_1

{

public class B

{

// Property

public String PropertyB { get; set; }

// Constructor

public B(String propertyB)

{

PropertyB = propertyB;

}

}

}

using Ex\_1\_1;

// Однонаправленная ассоциация от класса A к классу B кратности «один к одному».

internal class Program

{

public static void Main(String[] args)

{

A objectA = new A("class A");

B objectB = new B("class B");

// Ассоциация от класса A к классу B

objectA.ObjectB = objectB;

Console.WriteLine(objectA.ObjectB.PropertyB);

}

}

Результат работы программы представлен на рисунке 1. На рисунке 2 представлена UML-диаграмма классов для первого пункта эксперимента №1.

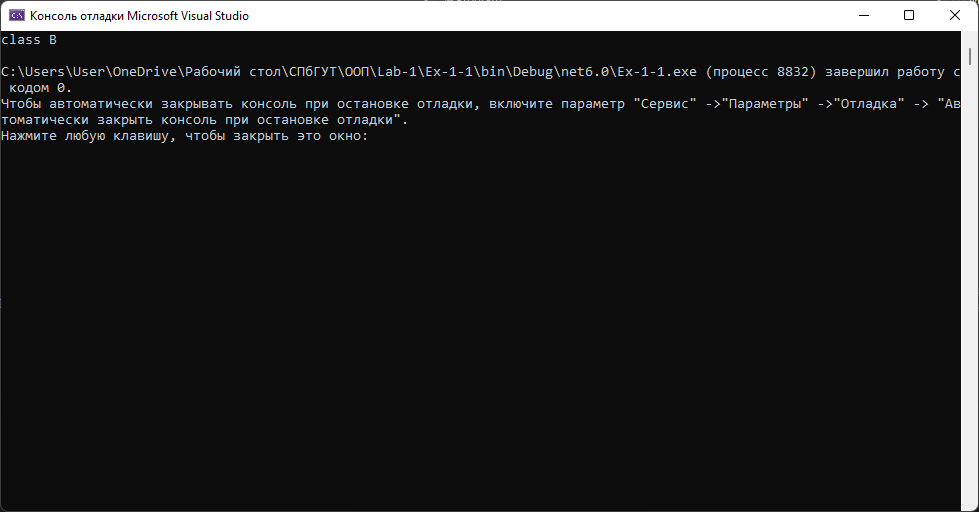


Рисунок 1 - результат работы программы

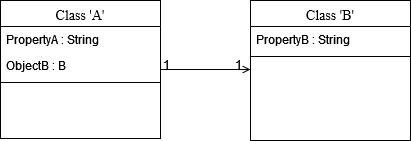
**

Рисунок 2 - UML-диаграмма (однонаправленная ассоциация кратности «один к одному»)

Аналогично первому заданию были созданы два класса A и B со свойствами PropertyA и PropertyB соответственно. В классе A была инициализирована ссылка в класс B, и наоборот, в классе B – в класс A. Таким образом, была установлена двунаправленная ассоциация кратности «один к одному». В теле метода Main описывается обращение из объекта класса A к свойству PropertyB объекта класса B. Исходный код двух классов и метода Main представлен ниже.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_2

{

public class A

{

// Properties

public String PropertyA { get; set; }

public B? ObjectB { get; set; }

// Constructor

public A(String propertyA)

{

PropertyA = propertyA;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_2

{

public class B

{

// Properties

public String PropertyB { get; set; }

public A? ObjectA { get; set; }

// Constructor

public B(String propertyB)

{

PropertyB = propertyB;

}

}

}

using Ex\_1\_2;

// Двунаправленная ассоциация кратности «один к одному»

internal class Program

{

public static void Main(String[] args)

{

A objectA = new A("class A");

B objectB = new B("class B");

objectA.ObjectB = objectB;

objectB.ObjectA = objectA;

// Обращение из объекта класса A, к свойству PropertyB объекта класса B.

Console.WriteLine(objectA.ObjectB.PropertyB);

// Обращение из объекта класса B, к свойству PropertyA объекта класса A.

Console.WriteLine(objectB.ObjectA.PropertyA);

}

}

Результат работы программы представлен на рисунке 3. На рисунке 4 представлена UML-диаграмма классов для второго пункта эксперимента №1.

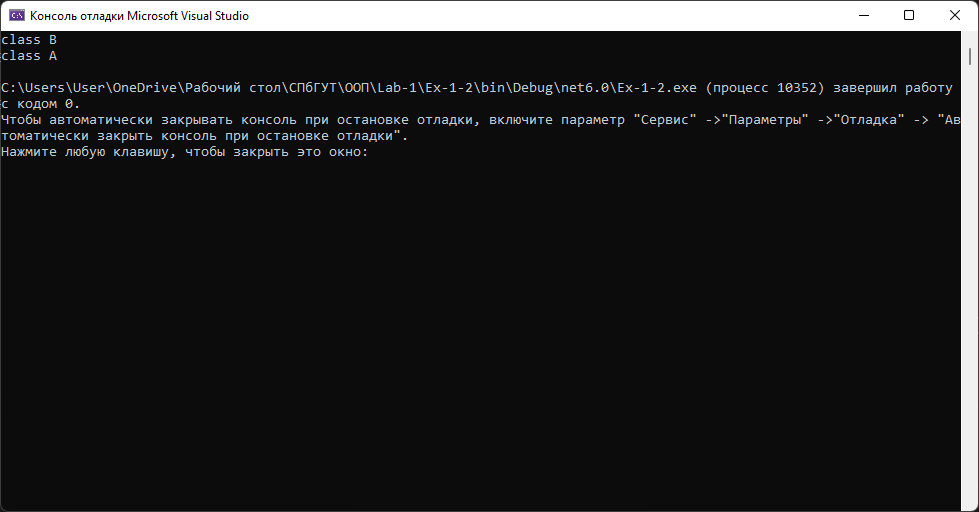


Рисунок 3 - результат работы программы



Рисунок 4 - UML-диаграмма (двунаправленная ассоциация кратности «один к одному»)

Реализация отношения агрегации между классами A и B осуществлялась через добавление ссылки на объект класса B в объект класса A (корень агрегата) c помощью метода Add(B b). В теле метода Main описывается обращение из объекта класса A к свойству PropertyB объекта класса B. Исходный код двух классов и метода Main представлен на следующих страницах (стр. 9-10).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_3

{

internal class A

{

// Properties

public String PropertyA { get; set; }

public List<B> ListB { get; set; }

// Constructor

public A(String propertyA)

{

PropertyA = propertyA;

ListB = new List<B>();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_3

{

internal class B

{

// Property

public String PropertyB { get; set; }

// Constructor

public B(String propertyB)

{

PropertyB = propertyB;

}

}

}

using Ex\_1\_3;

internal class Program

{

public static void Main(String[] args)

{

A objectA = new A("class A");

B objectB = new B("class B");

// Реализация отношения агрегации между классами A и B

objectA.ListB.Add(objectB);

Console.WriteLine(objectA.ListB.ElementAt(0).PropertyB);

}

}

Результат работы программы представлен на рисунке 5. На рисунке 6 представлена UML-диаграмма классов для третьего пункта эксперимента №1.

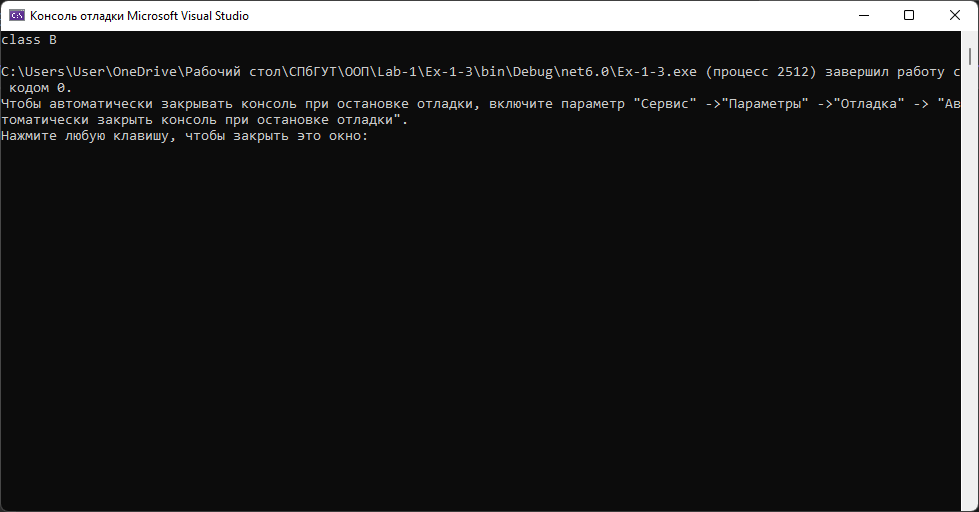


Рисунок 5 - результат работы программы

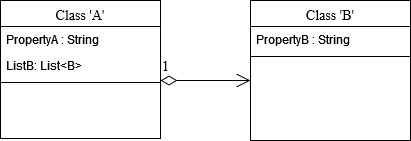


Рисунок 6 - UML-диаграмма (отношение агрегации кратности "один к одному")

Реализация композиции один ко многим осуществляется с помощью объявления ObjectB, экземпляра класса B, в конструкторе класса A. Исходный код двух классов и метода Main представлен ниже.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_4

{

public class A

{

// Properties

public String PropertyA { get; set; }

public B ObjectB { get; }

// Constructor

public A(String propertyA)

{

PropertyA = propertyA;

// Реализация отношения композиции между классами A и B

ObjectB = new B("B in A");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_1\_4

{

public class B

{

// Property

public String PropertyB { get; set; }

// Constructor

public B(String propertyB)

{

PropertyB = propertyB;

}

}

}

В методе Main описывается обращение из объекта класса A к свойству PropertyB объекта класса B.

using Ex\_1\_4;

internal class Program

{

public static void Main(String[] args)

{

A objectA = new A("class A");

Console.WriteLine(objectA.ObjectB.PropertyB);

}

}

Результат работы программы представлен на рисунке 7. На рисунке 8 представлена UML-диаграмма классов для четвёртого пункта эксперимента №1.

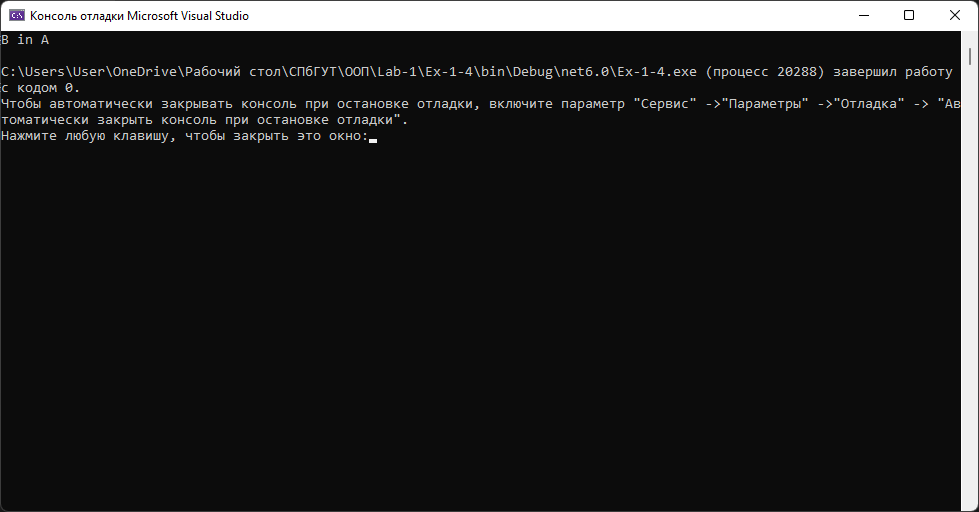


Рисунок 7 - результат работы программы

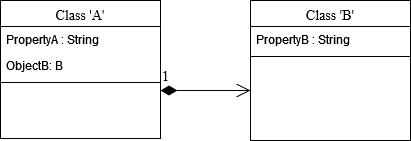


Рисунок 8 - UML-диаграмма (отношение композиции кратности "один к одному")

**Эксперимент 2**

Согласно 8 варианту созданы классы «Структурное подразделение» и «Сотрудник» и реализована однонаправленная ассоциация от основного класса к зависимому классу кратности «один ко многим». Класс «Структурное подразделение» (StructuralUnit) имеет свойство UnitNumber, а класс «Сотрудник» (Employee) - свойства Id и Name. Для каждого класса переопределен метод ToString(), который выводит описание состояния объекта в виде строки. В методе Main созданы один экземпляр основного класса и пять объектов, являющихся экземплярами зависимого класса. В консоль выводятся результаты вызова метода основного класса ToString(). Исходный код двух классов и метода Main представлен на следующих страницах (стр. 15-16).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_2

{

// Main class

internal class StructuralUnit

{

// Properties

public int UnitNumber { get; set; }

// Пять экземпляров зависимого класса

public Employee Employee1 { get; set; }

public Employee Employee2 { get; set; }

public Employee Employee3 { get; set; }

public Employee Employee4 { get; set; }

public Employee Employee5 { get; set; }

// Constructor

public StructuralUnit(int unitNumber, Employee e1, Employee e2, Employee e3, Employee e4, Employee e5)

{

UnitNumber = unitNumber;

Employee1 = e1;

Employee2 = e2;

Employee3 = e3;

Employee4 = e4;

Employee5 = e5;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Структурное подразделение={\n" + "UnitNumber=" + UnitNumber +

",\nСотрудник\_1=" + Employee1 + ",\nСотрудник\_2=" + Employee2 +

",\nСотрудник\_3=" + Employee3 + ",\nСотрудник\_4=" + Employee4 +

",\nСотрудник\_5=" + Employee5 + "\n}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_2

{

// Dependent class

internal class Employee

{

// Properties

public int Id { get; set; }

public String? Name { get; set; }

public Employee(int id, String name)

{

Id = id;

Name = name;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Сотрудник={Id=" + Id + ", Name=" + Name + "}";

}

}

}

/\* Однонаправленная ассоциация кратности "один ко многим"

Вариант 8: структурное подразделение (осн.класс) - сотрудник (зависимый класс) \*/

using Ex\_2;

internal class Program

{

public static void Main(String[] args)

{

Employee employee1 = new Employee(1, "Сотрудник\_1");

Employee employee2 = new Employee(2, "Сотрудник\_2");

Employee employee3 = new Employee(3, "Сотрудник\_3");

Employee employee4 = new Employee(4, "Сотрудник\_4");

Employee employee5 = new Employee(5, "Сотрудник\_5");

StructuralUnit unit = new StructuralUnit(1, employee1, employee2, employee3, employee4, employee5);

Console.WriteLine(unit.ToString());

}

}

Результат работы программы представлен на рисунке 9. На рисунке 10 представлена UML-диаграмма классов для эксперимента №2.

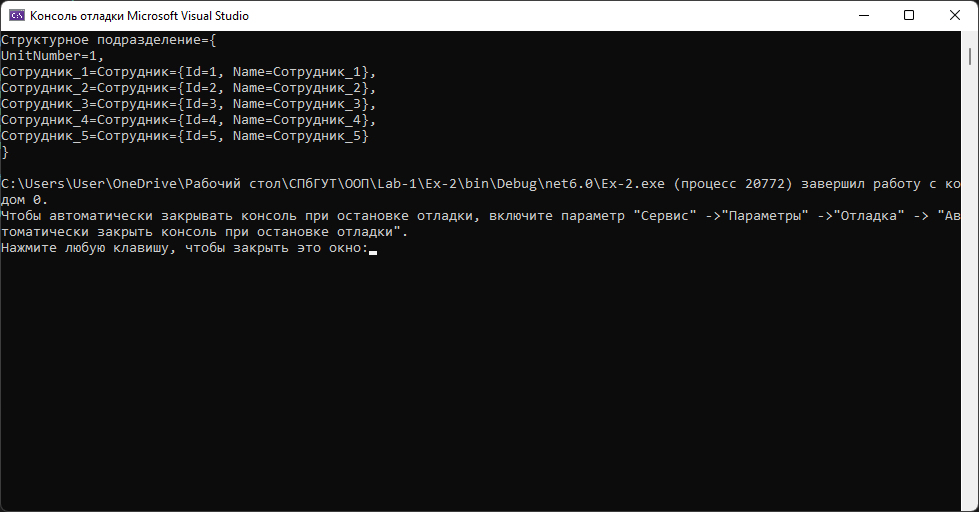


Рисунок 9 - результат работы программы

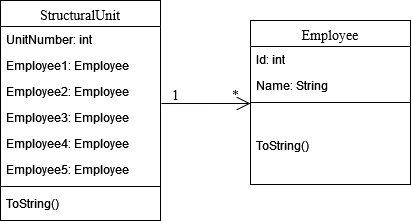


Рисунок 10 - UML-диаграмма (отношение ассоциации кратности "один ко многим")

**Эксперимент 3**

Для реализации двунаправленной ассоциации кратности «один ко многим» в зависимый класс из эксперимента №2 была добавлена обратная ссылка. Созданы один экземпляр основного класса и три экземпляра зависимого класса. В методе Main вызван метод основного класса ToString(), а в консоль выводятся результаты его работы, а затем описано обращение из объекта зависимого класса к основному с помощью обратной ссылки и вызван метод ToString(). Исходный код двух классов и метода Main представлен ниже.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_3

{

// Main class

internal class StructuralUnit

{

// Properties

public int UnitNumber { get; set; }

// Ссылка на коллекцию объектов зависимого класса

public Employee? Employee1 { get; set; }

public Employee? Employee2 { get; set; }

public Employee? Employee3 { get; set; }

// Constructor

public StructuralUnit(int unitNumber)

{

UnitNumber = unitNumber;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Структурное подразделение={" + "UnitNumber=" + UnitNumber + "}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_3

{

// Dependent class

internal class Employee

{

// Properties

public int Id { get; set; }

public String? Name { get; set; }

public StructuralUnit Unit { get; set; }

public Employee(int id, String name, StructuralUnit unit)

{

Id = id;

Name = name;

Unit = unit;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Сотрудник={Id=" + Id + ", Name=" + Name + ", Unit=" + Unit + "}";

}

}

}

/\* Двунаправленная ассоциация кратности "один ко многим"

Вариант 8: структурное подразделение (осн.класс) - сотрудник (зависимый класс) \*/

using Ex\_3;

internal class Program

{

public static void Main(String[] args)

{

StructuralUnit unit = new StructuralUnit(1);

Employee employee1 = new Employee(1, "Сотрудник\_1", unit);

Employee employee2 = new Employee(2, "Сотрудник\_2", unit);

Employee employee3 = new Employee(3, "Сотрудник\_3", unit);

unit.Employee1 = employee1;

unit.Employee2 = employee2;

unit.Employee3 = employee3;

// Вызов метода основного класса ToString()

Console.WriteLine(unit.ToString() + "\n");

// Обращение из объекта зависимого класса к основному с помощью обратной ссылки и вызов метода ToString()

Console.WriteLine(employee1.Unit.ToString());

Console.WriteLine(employee2.Unit.ToString());

Console.WriteLine(employee3.Unit.ToString() + "\n");

// Вывод метода зависимого класса ToString()

Console.WriteLine(employee1.ToString());

Console.WriteLine(employee2.ToString());

Console.WriteLine(employee3.ToString());

}

}

Результат работы программы представлен на рисунке 11. На рисунке 12 представлена UML-диаграмма классов для эксперимента №3.

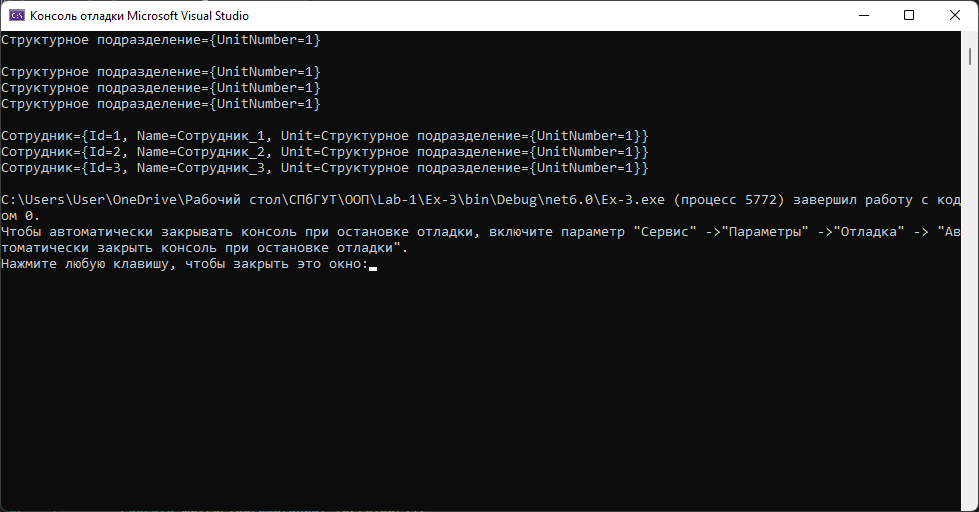


Рисунок 11 - результат работы программы

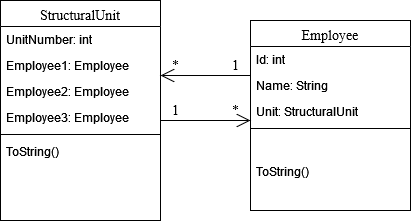


Рисунок 12 - двунаправленная ассоциация кратности "один ко многим"

**Эксперимент 4**

В ходе выполнения четвертого эксперимента для классов, созданных в результате эксперимента №2, реализовано отношение агрегации. В методе Main созданы один экземпляр основного класса (корня агрегата) и три экземпляра зависимого класса. Последовательно добавлены с помощью метода Add() корня агрегата ссылки на объекты зависимого класса. Вызван метод основного класса ToString() и выведены его результаты в консоль. Исходный код двух классов и метода Main представлены на следующих страницах (стр. 21-22).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_4

{

// Main class

internal class StructuralUnit

{

// Properties

public int UnitNumber { get; set; }

// Ссылка на коллекцию объектов зависимого класса

public List<Employee> UnitStaff { get; set; }

// Constructor

public StructuralUnit(int unitNumber)

{

UnitNumber = unitNumber;

UnitStaff = new List<Employee>();

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Структурное подразделение={\n" + "UnitNumber=" + UnitNumber + ",\nСотрудники=[" + String.Join(", ", UnitStaff) + "]\n}";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_4

{

// Dependent class

internal class Employee

{

// Properties

public int Id { get; set; }

public String? Name { get; set; }

public Employee(int id, String name)

{

Id = id;

Name = name;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Сотрудник={Id=" + Id + ", Name=" + Name + "}";

}

}

}

// Отношение агрегации «один ко многим»

using Ex\_4;

internal class Program

{

public static void Main(String[] args)

{

StructuralUnit unit = new StructuralUnit(1);

for (int i = 0; i < 3; i++)

{

unit.UnitStaff.Add(new Employee(i, "Name\_" + i));

}

Console.WriteLine(unit.ToString());

}

}

Результат работы программы представлен на рисунке 13. На рисунке 14 представлена UML-диаграмма классов для эксперимента №4.

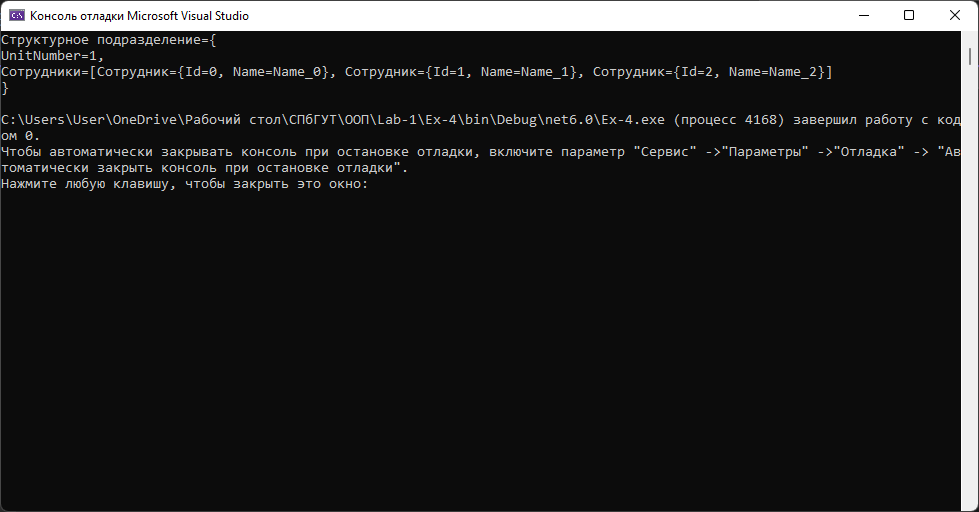


Рисунок 13 - результат работы программы

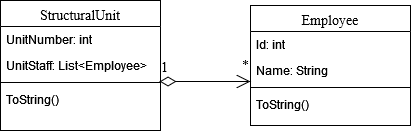


Рисунок 14 - отношение агрегации кратности один ко многим

**Эксперимент 5**

В ходе выполнения пятого эксперимента для классов, созданных в результате эксперимента №2, реализовано отношение композиции. В методе Main созданы один экземпляр основного класса и три экземпляра зависимого класса путем вызова метода New(), в который передаются в качестве параметров значения свойств объектов зависимого класса. Вызван метод основного класса ToString() и выведены его результаты в консоль. Исходный код двух классов и метода Main представлены на следующих страницах (стр. 24-25).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_5

{

// Main class

internal class StructuralUnit

{

// Properties

public int UnitNumber { get; set; }

// Пять экземпляров зависимого класса

public Employee Employee1 { get; set; }

public Employee Employee2 { get; set; }

public Employee Employee3 { get; set; }

// Constructor

public StructuralUnit(int unitNumber, int id1, String name1, int id2, String name2, int id3, String name3)

{

UnitNumber = unitNumber;

Employee1 = new Employee(id1, name1);

Employee2 = new Employee(id2, name2);

Employee3 = new Employee(id3, name3);

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Структурное подразделение={\n" + "UnitNumber=" + UnitNumber +

",\nСотрудник\_1=" + Employee1 + ",\nСотрудник\_2=" + Employee2 +

",\nСотрудник\_3=" + Employee3 + "\n}";

}

}

}

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_5

{

// Dependent class

internal class Employee

{

// Properties

public int Id { get; set; }

public String? Name { get; set; }

public Employee(int id, String name)

{

Id = id;

Name = name;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Сотрудник={Id=" + Id + ", Name=" + Name + "}";

}

}

}

// Отношение композиции «один ко многим»

using Ex\_5;

internal class Program

{

public static void Main(String[] args)

{

StructuralUnit unit = new StructuralUnit(1, 1, "Сотрудник\_1", 2, "Сотрудник\_2", 3, "Сотрудник\_3");

Console.WriteLine(unit.ToString());

}

}

Результат работы программы представлен на рисунке 15. На рисунке 16 представлена UML-диаграмма классов для эксперимента № 5.

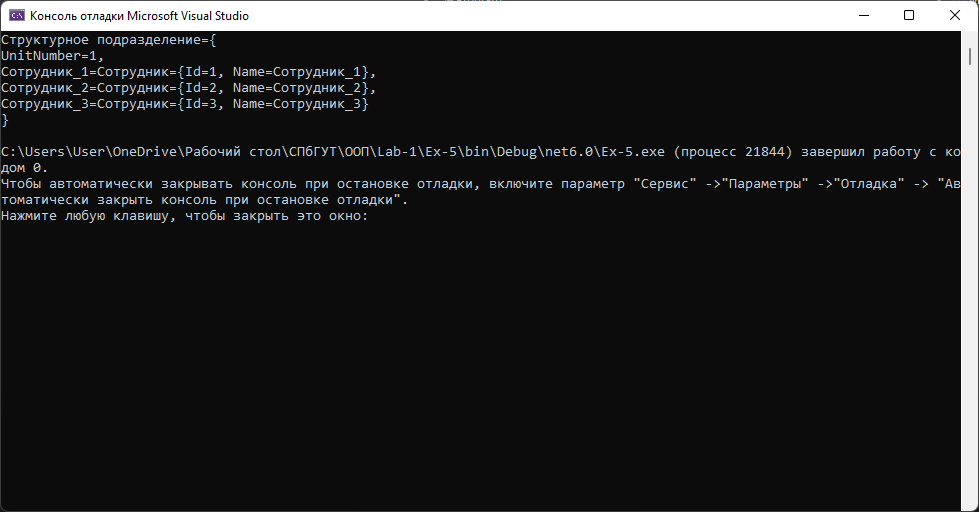


Рисунок 15 - результат работы программы

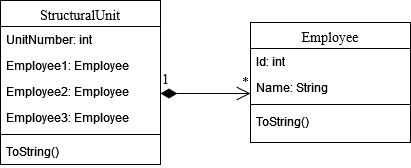


Рисунок 16 - UML-диаграмма (отношение композиции кратности "один ко многим")

**Эксперимент 6**

В ходе выполнения шестого эксперимента для классов, созданных в результате эксперимента №4, реализован метод глубокого клонирования, в котором создается клон (копия) объекта, на котором вызывается данный метод, при этом создаются копии всех агрегатов (с помощью цикла foreach). В методе Main созданы один экземпляр основного класса (корня агрегата) и три экземпляра зависимого класса. Последовательно (с помощью цикла for) добавлены с помощью метода Add() корня агрегата ссылки на объекты зависимого класса. Создан клон агрегата, путем вызова метода клонирования на корне агрегата. Вызван метод ToString() на корне агрегата и выведены результаты его работы в консоль. Исходный код двух классов и метода Main представлен на следующих страницах (стр. 27-28).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_6

{

// Main class

internal class StructuralUnit

{

// Properties

public int UnitNumber { get; set; }

// Ссылка на коллекцию объектов зависимого класса

public List<Employee> UnitStaff { get; set; }

// Constructor

public StructuralUnit(int unitNumber)

{

UnitNumber = unitNumber;

UnitStaff = new List<Employee>();

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Структурное подразделение={\n" + "UnitNumber=" + UnitNumber + ",\nСотрудники=[" + String.Join(", ", UnitStaff) + "]\n}";

}

public StructuralUnit Clone()

{

StructuralUnit unit = new StructuralUnit(UnitNumber);

foreach (Employee employee in UnitStaff)

{

unit.UnitStaff.Add(employee.Clone());

}

return unit;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Ex\_6

{

// Dependent class

internal class Employee

{

// Properties

public int Id { get; set; }

public String Name { get; set; }

public Employee(int id, String name)

{

Id = id;

Name = name;

}

// Переопределённый метод ToString() для вывода описания состояния объекта в виде строки

public override string ToString()

{

return "Сотрудник={Id=" + Id + ", Name=" + Name + "}";

}

public Employee Clone()

{

return new Employee(Id, Name);

}

}

}

// Глубокое клонирование объекта

using Ex\_6;

internal class Program

{

public static void Main(String[] args)

{

StructuralUnit unit = new StructuralUnit(1);

for (int i = 0; i < 3; i++)

{

unit.UnitStaff.Add(new Employee(i, "Name\_" + i));

}

Console.WriteLine(unit.ToString() + "\n");

// Клонирование объекта

StructuralUnit unitClone = unit.Clone();

// Вызов метода ToString() на корне агрегата

Console.WriteLine(unitClone.ToString());

}

}

Результат работы программы представлен на рисунке 17. На рисунке 18 представлена UML-диаграмма классов для эксперимента №5.

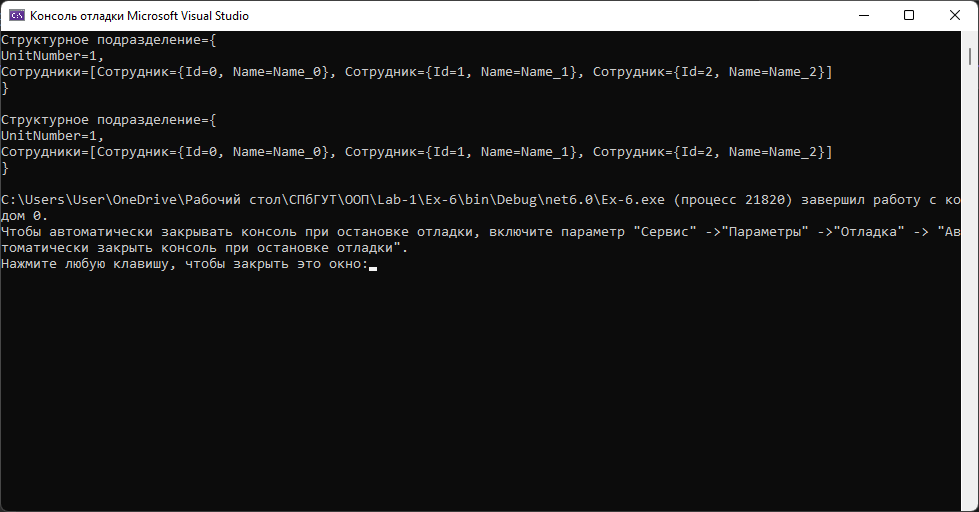


Рисунок 17 - результат работы программы

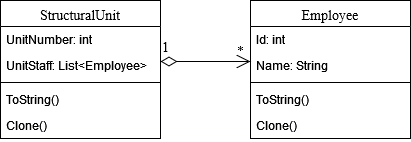


Рисунок 18 - UML-диаграмма (отношение агрегации кратности "один ко многим"); у каждого класса инициализирован метод клонирования

**Вывод**

Цель была достигнута и все задачи были выполнены. Реализация различных отношений (ассоциации, агрегации и композиции) разной кратности («один к одному» и «один ко многим») была закреплена на практике. В ходе выполнения лабораторной работы выяснилось, что построение UML-диаграмм – удобный и наглядный способ продемонстрировать взаимосвязь между классами.

**Список используемых источников**

* <https://habr.com/ru/post/458680/> - «Зачем нам UML? Или как сохранить себе нервы и время»
* <https://habr.com/ru/post/150041/> - «Отношение классов – от UML к коду»
* <https://metanit.com/sharp/patterns/1.2.php> - «Отношения между классами и объектами»