**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

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

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по лабораторной работе №1 **«АССОЦИАЦИЯ, АГРЕГАЦИЯ И**

**КОМПОЗИЦИЯ»**

по дисциплине «**Предметно-ориентированное проектирование автоматизированных систем управления**»

Выполнил: студент группы ИБ-32ВП, Ворожцов Г.С.

«29» октября 2025 г. \_\_\_\_\_\_\_\_\_\_\_/ Ворожцов Г.С. /

Принял: с. п. кафедры САР, А.П. Шабанов

«29» октября 2025 г. \_\_\_\_\_\_\_\_\_\_\_/ А.П. Шабанов

**Цель работы**

Цель работы: приобретение навыков реализации отношений между

**Эксперимент №1. Ассоциация, агрегация и композиция кратности**

«один к одному»

Задание для всех, без вариантов. Создать два класса A и B, имеющие

свойства PropertyA и PropertyB, соответственно.

1. Реализовать между данными классами однонаправленную

ассоциацию от класса A к классу B кратности «один к одному».

Создать экземпляры классов A и B. Установить между ними

ассоциацию. Обратиться из объекта класса A, к свойству PropertyB

объекта класса B. Проанализировать результат.

1. Реализовать между данными классами двунаправленную

ассоциацию кратности «один к одному», путем добавления

обратной ссылки (свойства навигации) в класс B. Создать

экземпляры классов A и B. Установить между двунаправленную

ними ассоциацию. Обратиться из объекта класса A, к свойству

PropertyB объекта класса B. Обратиться из объекта класса B, к

свойству PropertyA объекта класса A. Проанализировать результат.

1. 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. Проанализировать

результат.

1. Реализовать отношение композиции между классами A и B. Создать

экземпляр класса A. Обратиться из объекта класса A, к свойству

PropertyB объекта класса B. Проанализировать результат.

**Эксперимент №2. Однонаправленная ассоциация «один ко многим»**

Согласно своему варианту создать два класса 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

1

класса ToString(), вывести его результаты в консоль и проанализировать

результат его работы.

Варианты заданий

№ Основной класс Зависимый класс

4 Группа Студент

**Эксперимент №3. Двунаправленная ассоциация «один ко многим»**

В зависимый класс из предыдущего эксперимента 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().

**Эксперимент №4. 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 и проанализировать результат его работы.

**Эксперимент №5. 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

2

класса (объекты зависимых классов создаются внутри метода New()).Вызвать

метод основного класса ToString(), вывести его результаты в консоль и

проанализировать результат его работы.

**Эксперимент №6. 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.

Создать один экземпляр основного класса и три экземпляра зависимого

класса путем вызова метода New(), в который передаются в качестве

параметров значения свойств объектов зависимого класса (объекты

зависимых классов создаются внутри метода New()).Вызвать метод

основного класса ToString() и проанализировать результат его работы.

**Результат работы**

**EX1**

*EX1/Program.cs*

|  |
| --- |
| ﻿namespace App;    internal static class Program  {  public static void Main()  {  Demo1();  Demo2();  Demo3();  Demo4();  }    private static void Demo1()  {  Console.WriteLine("1) Ассоциация (однонаправленная)");  var a1 = new A1();  var b1 = new B1 {PropertyB = "test"};  a1.SetAssociation(b1);  Console.WriteLine(a1.B.PropertyB);  Console.WriteLine();  }    private static void Demo2()  {  Console.WriteLine("2) Ассоциация (двунаправленная)");  var a2 = new A2 {PropertyA = "test A"};  var b2 = new B2 {PropertyB = "test B"};  a2.SetAssociation(b2);  Console.WriteLine(a2.B.PropertyB);  Console.WriteLine(a2.B.A.PropertyA);  Console.WriteLine();  }    private static void Demo3()  {  Console.WriteLine("3) Агрегация 1->1");  var a = new A3();  var b = new B3 { PropertyB = "B3 value" };  a.Attach(b);  Console.WriteLine(a.B.PropertyB);  a.Dispose();  Console.WriteLine(b.PropertyB);  Console.WriteLine(b.Owner == null ? "да" : "нет");  Console.WriteLine();  }    private static void Demo4()  {  Console.WriteLine("4) Композиция 1->1");  B4 bRef;  using (var a = new A4("B4 value"))  {  bRef = a.B;  Console.WriteLine(a.B.PropertyB);  }    try  {  Console.WriteLine(bRef.PropertyB);  Console.WriteLine("сюда не доходим");  }  catch (ObjectDisposedException)  {  Console.WriteLine("ObjectDisposedException, доступа нет");  }  Console.WriteLine();  }  } |

*EX1/A1.cs*

|  |
| --- |
| namespace App;    public class A1  {  private B1? \_b1;  public B1 B => \_b1 ?? throw new Exception("B1 не инициализрована");    public void SetAssociation(B1 b)  {  if (\_b1 != null) throw new InvalidOperationException("Связь уже установлена.");  \_b1 = b ?? throw new ArgumentNullException(nameof(b));  }  } |

*EX1/A2.cs*

|  |
| --- |
| namespace App;    public class A2  {  public string PropertyA { get; set; } = "";    private B2? \_b2;    public B2 B => \_b2 ?? throw new Exception("B2 не инициализрована");    public void SetAssociation(B2 b)  {  if (\_b2 != null) throw new InvalidOperationException("Связь уже установлена.");  \_b2 = b ?? throw new ArgumentNullException(nameof(b));  \_b2.SetAssociation(this);  }  } |

*EX1/A3.cs*

|  |
| --- |
| namespace App;    public class A3 : IDisposable  {  private B3? \_b3;  public B3 B => \_b3 ?? throw new Exception("B3 не инициализрована");    public void Attach(B3 b)  {  if (b is null) throw new ArgumentNullException(nameof(b));  if (\_b3 != null) throw new InvalidOperationException("У A3 уже есть часть.");  if (b.Owner != null) throw new InvalidOperationException("B3 уже присоединён к другому A3.");  \_b3 = b;  b.Owner = this;  }    public B3 Detach()  {  var b = \_b3 ?? throw new InvalidOperationException("Нечего отсоединять.");  \_b3 = null;  b.Owner = null;  return b;  }    public void Dispose()  {  if (\_b3 != null)  {  \_b3.Owner = null;  \_b3 = null;  }  }  } |

*EX1/A4.cs*

|  |
| --- |
| namespace App;    public class A4: IDisposable  {  public B4 B { get; }    public A4(string valueForB) => B = new B4(valueForB);    public void Dispose()  {  B.Dispose();  }  } |

*EX1/B1.cs*

|  |
| --- |
| namespace App;    public class B1  {  public string PropertyB { get; set; } = "";  } |

*EX1/B2.cs*

|  |
| --- |
| namespace App;    public class B2  {  public string PropertyB { get; set; } = "";    private A2? \_a2;    public A2 A => \_a2 ?? throw new Exception("A2 не инициализрована");    public void SetAssociation(A2 a)  {  if (\_a2 != null) throw new InvalidOperationException("Связь уже установлена.");  \_a2 = a ?? throw new ArgumentNullException(nameof(a));  }  } |

*EX1/B3.cs*

|  |
| --- |
| namespace App;    public class B3  {  public string PropertyB { get; set; } = "";  public A3? Owner { get; internal set; }  } |

*EX1/B4.cs*

|  |
| --- |
| namespace App;    public class B4: IDisposable  {  private readonly string \_value;  private bool \_disposed;    internal B4(string value) => \_value = value;    public string PropertyB  {  get  {  if (\_disposed) throw new ObjectDisposedException(nameof(B4));  return \_value;  }  }    public void Dispose() => \_disposed = true;  } |

**EX2**

*EX2/Group.cs*

|  |
| --- |
| using System.Text;    namespace App;    public class Group  {  public string Name { get; }    private readonly List<Student> \_students = new();  public IReadOnlyList<Student> Students => \_students;    public Group(string name)  {  Name = name ?? throw new ArgumentNullException(nameof(name));  }    public void AddStudent(Student student)  {  if (student is null) throw new ArgumentNullException(nameof(student));  \_students.Add(student);  }    public override string ToString()  {  var sb = new StringBuilder();  sb.AppendLine($"Группа: \"{Name}\"");  sb.AppendLine($"Студентов: {Students.Count}");  for (int i = 0; i < \_students.Count; i++)  {  sb.AppendLine($" {i + 1}. {\_students[i]}");  }  return sb.ToString();  }  } |

*EX2/Student.cs*

|  |
| --- |
| namespace App;    public class Student  {  public string FirstName { get; }  public string LastName { get; }    public Student(string firstName, string lastName)  {  FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));  LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));  }    public override string ToString() => $"Студент(\"{FirstName} {LastName}\")";  } |

*EX2/Program.cs*

|  |
| --- |
| ﻿using App;    var group = new Group("Группа ПИ-101");    group.AddStudent(new Student("Иван", "Иванов"));  group.AddStudent(new Student("Мария", "Петрова"));  group.AddStudent(new Student("Сергей", "Смирнов"));  group.AddStudent(new Student("Анна", "Кузнецова"));  group.AddStudent(new Student("Дмитрий", "Соколов"));    Console.WriteLine(group.ToString()); |

**EX3**

*EX3/Group.cs*

|  |
| --- |
| using System.Text;    namespace App;    public class Group  {  public string Name { get; }    private readonly List<Student> \_students = new();  public IReadOnlyList<Student> Students => \_students;    public Group(string name)  {  Name = name ?? throw new ArgumentNullException(nameof(name));  }    public void AddStudent(Student student)  {  if (student is null) throw new ArgumentNullException(nameof(student));  if (!\_students.Contains(student))  {  \_students.Add(student);  student.Group = this;  }  }    public override string ToString()  {  var sb = new StringBuilder();  sb.AppendLine($"Группа: \"{Name}\"");  sb.AppendLine($"Студентов: {Students.Count}");  for (int i = 0; i < \_students.Count; i++)  {  sb.AppendLine($" {i + 1}. {\_students[i]}");  }  return sb.ToString();  }  } |

*EX3/Student.cs*

|  |
| --- |
| namespace App;    public class Student  {  public string FirstName { get; }  public string LastName { get; }    public Group? Group { get; internal set; }    public Student(string firstName, string lastName)  {  FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));  LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));  }    public override string ToString() => $"Студент(\"{FirstName} {LastName}\")";  } |

*EX3/Program.cs*

|  |
| --- |
| ﻿using App;    var group = new Group("Группа ПИ-201");    var s1 = new Student("Иван", "Иванов");  var s2 = new Student("Мария", "Петрова");  var s3 = new Student("Сергей", "Смирнов");    group.AddStudent(s1);  group.AddStudent(s2);  group.AddStudent(s3);    Console.WriteLine("Вывод группы через Group.ToString():");  Console.WriteLine(group.ToString());    Console.WriteLine("Вывод группы через обратную ссылку из объекта студента:");  Console.WriteLine(s2.Group?.ToString()); |

**EX4**

*EX4/Group.cs*

|  |
| --- |
| using System.Text;    namespace App;    public class Group  {  public string Name { get; }    private readonly List<Student> \_students = new();  public IReadOnlyList<Student> Students => \_students;    public Group(string name)  {  Name = name ?? throw new ArgumentNullException(nameof(name));  }    public void Add(Student student)  {  if (student is null) throw new ArgumentNullException(nameof(student));  if (!\_students.Contains(student))  \_students.Add(student);  }    public override string ToString()  {  var sb = new StringBuilder();  sb.AppendLine($"Группа: \"{Name}\"");  sb.AppendLine($"Студентов: {Students.Count}");  for (int i = 0; i < \_students.Count; i++)  {  sb.AppendLine($" {i + 1}. {\_students[i]}");  }  return sb.ToString();  }  } |

*EX4/Student.cs*

|  |
| --- |
| namespace App;    public class Student  {  public string FirstName { get; }  public string LastName { get; }    public Student(string firstName, string lastName)  {  FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));  LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));  }    public override string ToString() => $"Студент(\"{FirstName} {LastName}\")";  } |

*EX4/Program.cs*

|  |
| --- |
| ﻿using App;    var group = new Group("Группа АГ-301");    var s1 = new Student("Иван", "Иванов");  var s2 = new Student("Мария", "Петрова");  var s3 = new Student("Сергей", "Смирнов");    group.Add(s1);  group.Add(s2);  group.Add(s3);    Console.WriteLine(group.ToString()); |

**EX5**

*EX5/Group.cs*

|  |
| --- |
| using System.Text;    namespace App;    public class Group  {  public string Name { get; }    private readonly List<Student> \_students = new();  public IReadOnlyList<Student> Students => \_students;    public Group(string name)  {  Name = name ?? throw new ArgumentNullException(nameof(name));  }    public void New(string firstName, string lastName)  {  var student = new Student(firstName, lastName);  \_students.Add(student);  }    public override string ToString()  {  var sb = new StringBuilder();  sb.AppendLine($"Группа: \"{Name}\"");  sb.AppendLine($"Студентов: {Students.Count}");  for (int i = 0; i < \_students.Count; i++)  {  sb.AppendLine($" {i + 1}. {\_students[i]}");  }  return sb.ToString();  }  } |

*EX5/Student.cs*

|  |
| --- |
| namespace App;    public class Student  {  public string FirstName { get; }  public string LastName { get; }    public Student(string firstName, string lastName)  {  FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));  LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));  }    public override string ToString() => $"Студент(\"{FirstName} {LastName}\")";  } |

*EX5/Program.cs*

|  |
| --- |
| ﻿using App;    var group = new Group("Группа КОМ-401");    group.New("Иван", "Иванов");  group.New("Мария", "Петрова");  group.New("Сергей", "Смирнов");    Console.WriteLine(group.ToString()); |

**EX6**

*EX6/Group.cs*

|  |
| --- |
| using System.Text;    namespace App;    public class Group  {  public string Name { get; }    private readonly List<Student> \_students = new();  public IReadOnlyList<Student> Students => \_students;    public Group(string name)  {  Name = name ?? throw new ArgumentNullException(nameof(name));  }    public void Add(Student student)  {  if (student is null) throw new ArgumentNullException(nameof(student));  if (!\_students.Contains(student))  \_students.Add(student);  }    public void New(string firstName, string lastName)  {  var student = new Student(firstName, lastName);  \_students.Add(student);  }    public Group DeepClone()  {  var clone = new Group(Name);  foreach (var s in \_students)  {  clone.\_students.Add(new Student(s.FirstName, s.LastName));  }  return clone;  }    public override string ToString()  {  var sb = new StringBuilder();  sb.AppendLine($"Группа: \"{Name}\"");  sb.AppendLine($"Студентов: {Students.Count}");  for (int i = 0; i < \_students.Count; i++)  {  sb.AppendLine($" {i + 1}. {\_students[i]}");  }  return sb.ToString();  }  } |

*EX6/Student.cs*

|  |
| --- |
| namespace App;    public class Student  {  public string FirstName { get; }  public string LastName { get; }    public Student(string firstName, string lastName)  {  FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));  LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));  }    public override string ToString() => $"Студент(\"{FirstName} {LastName}\")";  } |

*EX6/Program.cs*

|  |
| --- |
| ﻿using App;    var groupAgg = new Group("Группа КЛ-501");    var a1 = new Student("Иван", "Иванов");  var a2 = new Student("Мария", "Петрова");  var a3 = new Student("Сергей", "Смирнов");    groupAgg.Add(a1);  groupAgg.Add(a2);  groupAgg.Add(a3);    var clone = groupAgg.DeepClone();    Console.WriteLine("Оригинал (агрегация):");  Console.WriteLine(groupAgg.ToString());  Console.WriteLine("Клон (глубокая копия):");  Console.WriteLine(clone.ToString());    var groupComp = new Group("Группа КЛ-502");  groupComp.New("Анна", "Кузнецова");  groupComp.New("Дмитрий", "Соколов");  groupComp.New("Ольга", "Ильина");    Console.WriteLine("Композиция (создание через New):");  Console.WriteLine(groupComp.ToString()); |

**Вывод**

В ходе выполнения лабораторной работы реализованы и проанализированы базовые отношения между классами: однонаправленная и двунаправленная ассоциации, агрегация, композиция, а также глубокое клонирование агрегата.