Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

Инструкция
по выполнению лабораторной работы №11
«Разработка, отладка и испытание программ, реализующих отношения между классами»

Минск 2024 г.

Лабораторная работа № 11

Тема работы: «Разработка, отладка и испытание программ, реализующих отношения между классами»

1 Цель работы

Сформировать умения разрабатывать алгоритмы и реализовать программы, реализующие отношения между классами.

2 Задание

Номер варианта соответствует номеру по списку в журнале.

1. Класс «КОМНАТА», содержит закрытые поля: сведения о метраже, высоте потолков и количестве окон и метод подсчета площади комнаты. Класс «КВАРТИРА», содержит закрытые поля: номер этажа, массив объектов класса «КОМНАТА», метод подсчета площади квартиры и метод вывода информации о комнатах квартиры.
2. Создать объект класса Автомобиль, используя классы Колесо, Двигатель. Методы: ехать, заправляться, менять колесо, вывести на консоль марку автомобиля.
3. Создать объект класса Самолет, используя классы Крыло, Шасси, Двигатель. Методы: летать, задавать маршрут, вывести на консоль маршрут.
4. Создать объект класса Государство, используя классы Область, Район, Город. Методы: вывести на консоль столицу, количество областей, площадь, областные центры.
5. Создать объект класса Планета, используя классы Материк, Океан, Остров. Методы: вы вести на консоль название материка, планеты, количество материков.
6. Создать объект класса Звездная система, используя классы Планета, Звезда, Луна. Методы: вывести на консоль количество планет в звездной системе, название звезды, добавление планеты в систему.
7. Создать объект класса Компьютер, используя классы Винчестер, Дисковод, Оперативная память, Процессор. Методы: включить, выключить, проверить на вирусы, вывести на консоль размер винчестера.
8. Создать объект класса Квадрат, используя классы Точка, Отрезок. Методы: задание размеров, растяжение, сжатие, поворот, изменение цвета.
9. Создать объект класса Круг, используя классы Точка, Окружность. Методы: задание размеров, изменение радиуса, определение принадлежности точки данному кругу.
10. Создать объект класса Щенок, используя классы Животное, Собака. Методы: вывести на консоль имя, подать голос, прыгать, бегать, кусать.
11. Создать объект класса Наседка, используя классы Птица, Кукушка. Методы: летать, петь, нести яйца, высиживать птенцов.

12. Создать объект класса Текстовый файл, используя классы Файл, Директория. Методы: создать, переименовать, вывести на консоль содержимое, дополнить, удалить.
13. Создать объект класса Одномерный массив, используя классы Массив, Элемент. Методы: создать, вывести на консоль, выполнить операции (сложить, вычесть, перемножить).
14. Создать объект класса Простая дробь, используя класс Число. Методы: вывод на экран, сложение, вычитание, умножение, деление.
15. Создать объект класса Дом, используя классы Окно, Дверь. Методы: закрыть на ключ, вывести на консоль количество окон, дверей.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

Наследование (inheritance) является одним из ключевых моментов ООП. Благодаря наследованию один класс может унаследовать функциональность другого класса.

Например, существует класс Person, который описывает отдельного человека:

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

Если необходим класс, описывающий сотрудника предприятия – создадим класс Employee. Поскольку этот класс будет реализовывать ту же функциональность, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником, или подклассом) от класса Person, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

```
class Employee : Person
{

}
```

После двоеточия мы указываем базовый класс для данного класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же свойства, методы, поля, которые есть в классе Person. Единственное, что не передается при наследовании, это конструкторы базового класса.

Таким образом, наследование реализует отношение is-a (является), объект класса Employee также является объектом класса Person:

```
static void Main(string[] args)
{
    Person p = new Person { Name = "Tom"};
    p.Display();
    p = new Employee { Name = "Sam" };
    p.Display();
    Console.Read();
}
```

Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений:

– Не поддерживается множественное наследование, класс может наследоваться только от одного класса.

– При создании производного класса надо учитывать тип доступа к базовому классу - тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public.

– Однако следует также учитывать, что если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производый класс может наследовать только от класса, который имеет модификатор public.

– Если класс объявлен с модификатором sealed, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников:

```
sealed class Admin
{
}
```

– Нельзя унаследовать класс от статического класса.

Пример реализации наследования классов.

```
using System;

public enum PublicationType { Misc, Book, Magazine, Article };

public abstract class Publication
{
   private bool published = false;
   private DateTime datePublished;
   private int totalPages;

   public Publication(string title, string publisher, PublicationType type)
   {
      if (String.IsNullOrWhiteSpace(publisher))
         throw new ArgumentException("The publisher is required.");
      Publisher = publisher;

      if (String.IsNullOrWhiteSpace(title))
         throw new ArgumentException("The title is required.");
      Title = title;

      Type = type;
   }

   public string Publisher { get; }
```

```
   public string Title { get; }

   public PublicationType Type { get; }

   public string CopyrightName { get; private set; }

   public int CopyrightDate { get; private set; }

   public int Pages
   {
     get { return totalPages; }
     set
     {
         if (value <= 0)
            throw new ArgumentOutOfRangeException("The number of pages cannot
be zero or negative.");
         totalPages = value;
     }
   }

   public string GetPublicationDate()
   {
      if (!published)
         return "NYP";
      else
         return datePublished.ToString("d");
   }

   public void Publish(DateTime datePublished)
   {
      published = true;
      this.datePublished = datePublished;
   }

   public void Copyright(string copyrightName, int copyrightDate)
   {
      if (String.IsNullOrWhiteSpace(copyrightName))
         throw new ArgumentException("The name of the copyright holder is
required.");
      CopyrightName = copyrightName;

      int currentYear = DateTime.Now.Year;
      if (copyrightDate < currentYear - 10 || copyrightDate > currentYear +
2)
         throw new ArgumentOutOfRangeException($"The copyright year must be
between {currentYear - 10} and {currentYear + 1}");
      CopyrightDate = copyrightDate;
   }

   public override string ToString() => Title;
}
```

Класс Book представляет книгу как специализированный тип публикации. В следующем примере показан исходный код класса Book.

```
using System;

public sealed class Book : Publication
{
   public Book(string title, string author, string publisher) :
          this(title, String.Empty, author, publisher)
   { }

   public Book(string title, string isbn, string author, string publisher) :
base(title, publisher, PublicationType.Book)
```

```
   {
      // isbn argument must be a 10- or 13-character numeric string without
"-" characters.
      // We could also determine whether the ISBN is valid by comparing its
checksum digit
      // with a computed checksum.
      //
      if (! String.IsNullOrEmpty(isbn)) {
        // Determine if ISBN length is correct.
        if (! (isbn.Length == 10 | isbn.Length == 13))
            throw new ArgumentException("The ISBN must be a 10- or 13-
character numeric string.");
        ulong nISBN = 0;
        if (! UInt64.TryParse(isbn, out nISBN))
            throw new ArgumentException("The ISBN can consist of numeric
characters only.");
      }
      ISBN = isbn;

      Author = author;
   }

   public string ISBN { get; }

   public string Author { get; }

   public Decimal Price { get; private set; }

   // A three-digit ISO currency symbol.
   public string Currency { get; private set; }

   // Returns the old price, and sets a new price.
   public Decimal SetPrice(Decimal price, string currency)
   {
       if (price < 0)
          throw new ArgumentOutOfRangeException("The price cannot be
negative.");
       Decimal oldValue = Price;
       Price = price;

       if (currency.Length != 3)
          throw new ArgumentException("The ISO currency symbol is a 3-
character string.");
       Currency = currency;

       return oldValue;
   }

   public override bool Equals(object obj)
   {
      Book book = obj as Book;
      if (book == null)
         return false;
      else
         return ISBN == book.ISBN;
   }

   public override int GetHashCode() => ISBN.GetHashCode();

   public override string ToString() => $"{(String.IsNullOrEmpty(Author) ? ""
: Author + ", ")}{Title}";
}
```

Помимо элементов, которые он наследует от Publication, класс Book определяет и переопределяет следующие члены.

Два конструктора Book используют три общих параметра. Два из них, header и publisher, соответствуют параметрам конструктора Publication. Третий — это author, который хранится в общедоступном неизменяемом свойстве Author. Один конструктор использует параметр isbn, который хранится в автосвойстве ISBN.

Первый конструктор использует ключевое слово this для вызова второго конструктора. Создание цепочки конструкторов — это обычный метод определения конструкторов. Конструкторы с меньшим числом параметров используют значения по умолчанию, вызывая конструкторы с большим числом параметров.

Второй конструктор использует ключевое слово base, чтобы передать заголовок и имя издателя в конструктор базового класса. Если вы не используете явный вызов конструктора базового класса в исходном коде, компилятор C# автоматически добавляет вызов конструктора по умолчанию (без параметров) для базового класса.

Свойство ISBN, доступное только для чтения, которое возвращает международный стандартный номер книги (уникальное 10- или 13-значное число) для объекта Book. Номер ISBN передается в качестве аргумента одному из конструкторов Book. Он сохраняется в закрытом резервном поле, автоматически создаваемым компилятором.

Свойство Author, доступное только для чтения. Имя автора передается в качестве аргумента обоим конструкторам Book и сохраняется в свойстве.

Два свойства, Price и Currency, с информацией о цене, доступные только для чтения. Значения этих свойств передаются в качестве аргументов при вызове метода SetPrice. Свойство Currency содержит трехзначное обозначение валюты по стандарту ISO (например, USD обозначает доллар США). Обозначение валюты по стандарту ISO можно получить из свойства ISOCurrencySymbol. Оба эти свойства доступны для чтения извне, но их можно задать в коде в классе Book.

Метод SetPrice, который задает значения для свойств Price и Currency. Эти значения возвращаются теми же свойствами.

Переопределение метода ToString, унаследованного от Publication, а также методов Object.Equals(Object) и GetHashCode, унаследованных от Object.

Виды отношений между классами

1. «Является» (IS-A)

С помощью диаграмм UML отношение между классами выражается в незакрашенной стрелочке от класса-наследника к классу-родителю (рисунок 1).

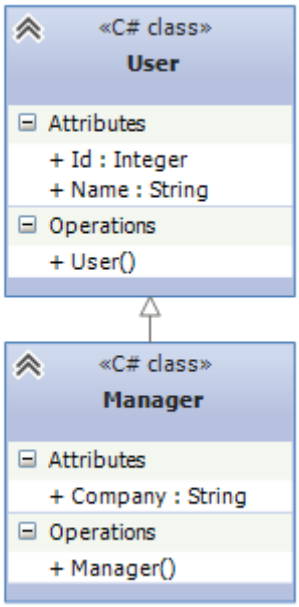

Рисунок 1 – Пример реализации отношения «является» с помощью диаграмм UML

2. «Реализация».

Реализация предполагает определение интерфейса и его реализация в классах. Например, имеется интерфейс IMovable с методом Move, который реализуется в классе Car:

```
public interface IMovable
{
    void Move();
}
public class Car : IMovable
{
    public void Move()
    {
        Console.WriteLine("Машина едет");
    }
}
```

С помощью диаграмм UML отношение реализации также выражается в незакрашенной стрелке от класса к интерфейсу, только линия теперь пунктирная (рисунок 2).

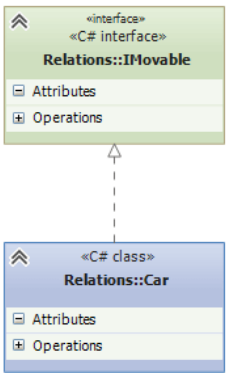

Рисунок 2 – Пример отношения «реализация» в диаграмме UML

3. «Ассоциация».

Ассоциация — это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Например, объект одного типа содержит или использует объект другого типа. Например, игрок играет в определенной команде:

```
class Team
{

}
class Player
{
    public Team Team { get; set; }
}
```

Класс Player связан отношением ассоциации с класом Team. На схемах UML ассоциация обозначается в виде обычно стрелки (рисунок 3).

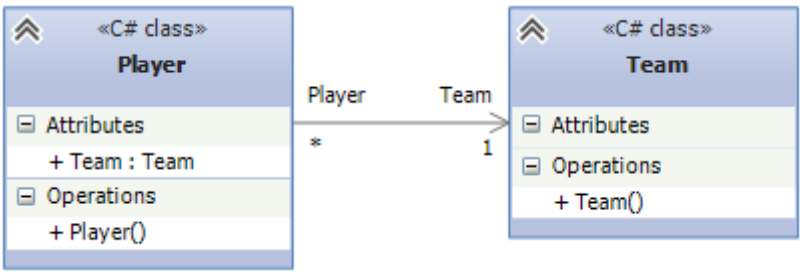

Рисунок 3 – Пример реализации отношения «ассоциация» с помощью диаграмм UML

Нередко при отношении ассоциации указывается кратность связей. В данном случае единица у Team и звездочка у Player на диаграмме отражает связь 1 ко многим. То есть, одна команда будет соответствовать многим игрокам.

Агрегация и композиция являются частными случаями ассоциации.

4. «Композиция» (Has-A)

Композиция определяет отношение HAS A, то есть отношение «имеет». Например, в класс автомобиля содержит объект класса электрического двигателя:

```
public class ElectricEngine
{ }

public class Car
{
    ElectricEngine engine;
    public Car()
    {
        engine = new ElectricEngine();
    }
}
```

На диаграммах UML отношение композиции проявляется в обычной стрелке от главной сущности к зависимой, при этом со стороны главной сущности, которая содержит, объект второй сущности, располагается закрашенный ромб (рисунок 4).

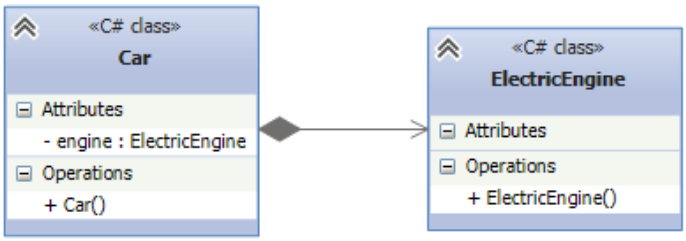

Рисунок 4 – Пример реализации отношения «композиция» с помощью диаграмм UML

5. «Агрегация» (Has-A).

От композиции следует отличать агрегацию. Она также предполагает отношение HAS A, но реализуется она иначе:

```
public abstract class Engine
{ }

public class Car
{
    Engine engine;
    public Car(Engine eng)
    {
        engine = eng;
    }
}
```

При агрегации реализуется слабая связь, то есть в данном случае объекты Car и Engine будут равноправны. В конструктор Car передается ссылка на уже имеющийся объект Engine. И, как правило, определяется ссылка не на конкретный класс, а на абстрактный класс или интерфейс, что увеличивает гибкость программы.

Отношение агрегации на диаграммах UML отображается также, как и отношение композиции, только теперь ромб будет незакрашенным (рисунок 5).

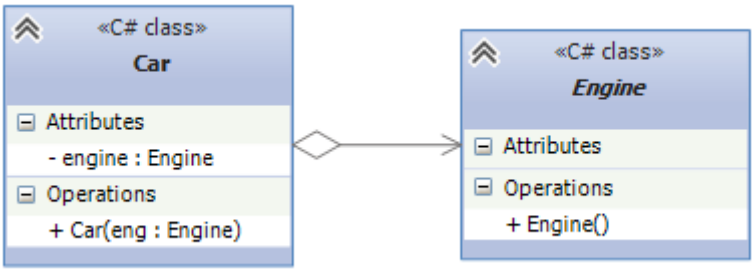

Рисунок 5 - Пример реализации отношения «агрегация» с помощью диаграмм UML

5. Порядок выполнения работы

1. Выделить ключевые моменты задачи.
2. Построить алгоритм и теоретическую объектную модель решения задачи.
3. Запрограммировать полученные алгоритмы и объектную модель.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Перечислите ограничения, которые нужно учитывать при наследовании классов.
2. Дайте определение понятию «ассоциация»? Поясните функциональное назначение.
3. Дайте определение понятию «агрегация»? Поясните функциональное назначение.
4. Дайте определение понятию «композиция»? Поясните функциональное назначение.

8. Рекомендуемая литература

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

2. Прайс, М. Дж. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

3. Васильев, А.Н. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

4. Фримен, А. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.