Тема: «Пространства имён. Система типов. Class. Struct. Enum. Record»

Цель: получить навыки создания и использования собственных пространств имён. Изучить основные способы организации пользовательских типов данных.

Время выполнения: 4 часа.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Пространства имён

Пространства имен активно используются в программировании на C# двумя способами.

Во-первых, .NET использует пространства имен для организации своих многочисленных классов следующим образом:

```
System.Console.WriteLine("Hello world");
```

`System` – это пространство имен, а `Console` – класс в этом пространстве имен. Ключевое слово `using` можно использовать, чтобы полное имя не требовалось, как в следующем примере:

```
using System;
Console.WriteLine("Hello world");
```

Во-вторых, объявление собственных пространств имен может помочь контролировать область действия имен классов и методов в более крупных программных проектах. Для объявления собственного пространства имён используется ключевое слово `namespace`. Пример создания своего пространства имён:

```
namespace MyNameSpace
{
    //TODO
}
```

Требования к пространствам имён:

1. Имя пространства имен должно быть допустимым именем идентификатора.

Начиная с C# 10, можно объявить пространство имен для всех типов, определенных в этом файле. Преимущество этого нового синтаксиса

заключается в том, что он проще, экономит горизонтальное пространство и фигурные скобки. Это облегчает чтение кода. Пример:

```
namespace MyNameSpace;
internal class A
{
    //TODO
}
internal class B
{
    //TODO
}
```

Пространства имен имеют следующие свойства:

- они организуют большие проекты по коду;
- они разграничиваются с помощью оператора *«.»* (точка);

Директива `using` избавляет от необходимости указывать имя пространства имен для каждого класса;

Пространство имен `global` является "корневым": `global::System` всегда будет ссылаться на пространство имен `System в .NET`.

Система типов

C# – это строго типизированный язык. Каждая переменная и константа имеет тип, как и каждое выражение, результатом вычисления которого является значение. Каждое объявление метода определяет имя и тип (значение, ссылка или вывод) для каждого входного параметра и возвращаемого значения. Библиотека классов .NET определяет встроенные числовые типы и сложные типы, представляющие широкий спектр конструкций. К ним относятся файловая система, сетевые подключения, коллекции и массивы объектов, а также даты. Типичная программа на C# использует типы из библиотеки классов и определяемые пользователем типы, которые моделируют концепции, специфичные для предметной области программы.

Информация, хранящаяся в типе, может включать следующие элементы:

- дисковое пространство, необходимое переменной типа.
- максимальное и минимальное значения, которые он может представлять;
- члены (методы, поля, события и т. д.), которые он содержит;
- базовый тип, от которого он наследуется;
- интерфейс(ы), которые он реализует;
- виды операций, которые разрешены.

Компилятор использует сведения о типах, чтобы убедиться, что все операции, выполняемые в коде, являются *типобезопасными*. Например, если объявить переменную типа `int`, компилятор позволяет использовать переменную в операциях сложения и вычитания. Если вы попытаетесь выполнить те же операции с переменной типа `bool`, компилятор выдаст ошибку. Пример:

```
int a = 2;
int b = a + 2;
//Output: 6
Console.WriteLine(a + b);
bool c = false;
//Output: Оператор "+" невозможно применить к операнду типа
"int" и "bool"
Console.WriteLine(a+c);
```

Встроенные типы

C# предоставляет стандартный набор встроенных типов. Они представляют целые числа, значения с плавающей запятой, логические выражения, текстовые символы, десятичные значения и другие типы данных. Также есть встроенные и типы. Эти типы можно использовать в любой программе на C#.

Пользовательские типы

Конструкции `struct`, `class`, `interface`, `enum` и `record` используются для создания собственных пользовательских типов. Сама библиотека классов .NET представляет собой коллекцию пользовательских типов, которые можно использовать в собственных приложениях. По умолчанию наиболее часто используемые типы в библиотеке классов доступны в любой программе на C#. Другие становятся доступными только при явном добавлении ссылки на проект в сборку, которая их определяет. После того как компилятор получит ссылку на сборку, можно объявить переменные (и константы) типов, объявленных в этой сборке в исходном коде.

Общая система типов

Важно понимать два фундаментальных момента, связанных с системой типов в .NET:

Платформа поддерживает принцип наследования. Типы могут быть производными от других типов, называемых *базовыми типами*. Производный тип наследует (с некоторыми ограничениями) методы, свойства и другие члены базового типа. Базовый тип, в свою очередь, может быть производным от какого-либо другого типа, и в этом случае

производный тип наследует члены обоих базовых типов в своей иерархии наследования. Все типы, включая встроенные числовые типы, такие как *System.Int32* в конечном итоге являются производными от одного базового типа, которым является *System.Object*. Эта унифицированная иерархия типов называется *Common Type System* (*CTS*).

Каждый тип в *CTS* определяется либо *как тип значения*, либо как *ссылочный тип*. К этим типам относятся все пользовательские типы в библиотеке классов .NET, а также пользовательские типы. Типы, определяемые с помощью ключевого слова *struct*, являются типами значений. Например, все встроенные числовые типы определены как `structs`. Если в определении типа используется ключевое слово `class` или *record*, он является ссылочным типом. Для ссылочных типов и типов значений используются разные правила компиляции, и они демонстрируют разное поведение во время выполнения.

На рисунке 1 показана связь между типами значений и ссылочными типами в CTS.

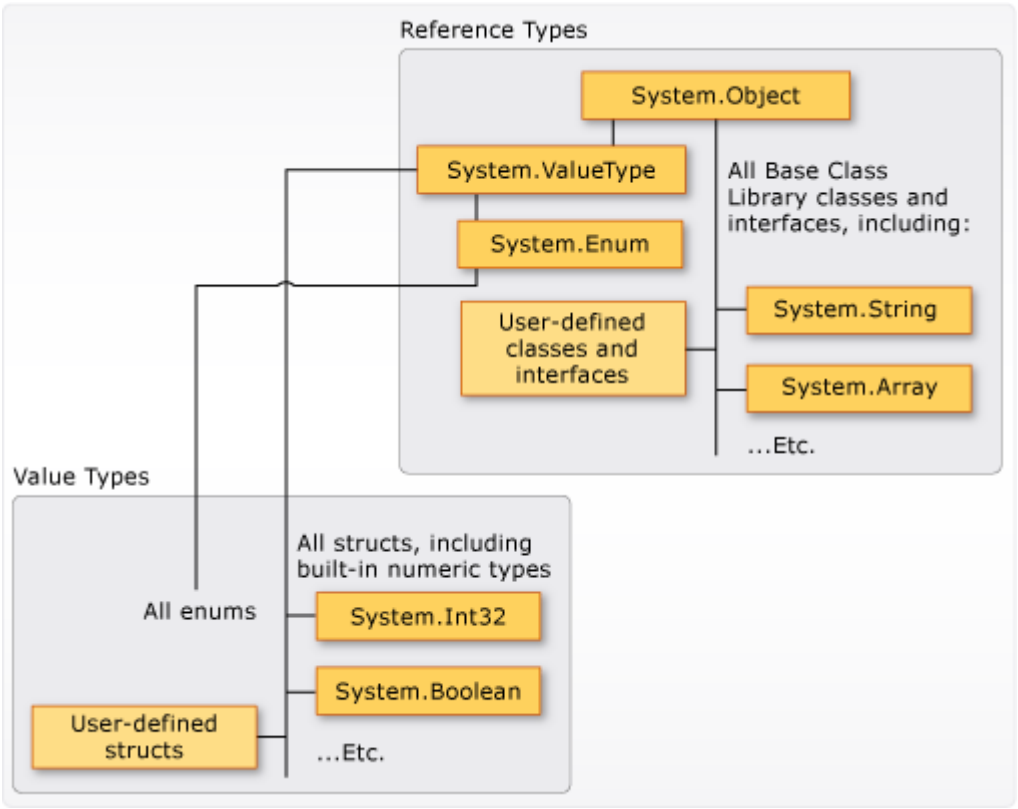


Рисунок 1 – Связь между ссылочными и типами значений

Классы и структуры являются двумя основными конструкциями системы общих типов CTS, используемой на платформе .NET. В C# 9

добавлены записи, которые представляют собой тип класса. Оба они являются структурами данных, которые инкапсулируют набор данных и поведений в одной логической сущности. Данные и поведение являются членами класса, структуры или записи. К ним относятся методы, свойства, события и другие элементы.

Объявление класса, структуры или записи представляет собой своего рода чертеж, на основе которого создаются экземпляры или объекты во время выполнения. Если вы определите класс, структуру или запись с именем `Person`, то `Person` здесь обозначает имя типа. Если вы объявите и инициализируете переменную p типа `Person`, принято говорить, что p является объектом (или экземпляром) `Person`. Можно создать несколько экземпляров одного типа `Person`, и каждый экземпляр будет иметь разные значения свойств и полей.

```
//Объяевление класса Person
internal class Person
{
    //TODO
}

//Объект класса Person
Person p = new Person();
```

Класс является ссылочным типом. Когда вы создаете объект и назначаете его переменной, эта переменная содержит только ссылку на память объекта. Если ссылка на объект сохраняется в новую переменную, эта переменная также ссылается на исходный объект. Изменения, внесенные через одну переменную, отражаются и в другой переменной, поскольку обе они ссылаются на одни и те же данные.

```
//Объекты класса Person
Person a = new Person(17);
Person b = new Person(18);

//Output: Age of person = 17
Console.WriteLine(a);
//Output: Age of person = 18
Console.WriteLine(b);

b.Age = 20;
a = b;
//Output: Age of person = 20
Console.WriteLine(a);
```

```
//Output: Age of person = 20
Console.WriteLine(b);
```

Структура (*struct*) является типом значения. При создании структуры переменная, которой присвоена структура, содержит фактические данные этой структуры. Если структура присваивается новой переменной, все данные копируются. Таким образом, новая переменная и исходная переменная содержат две отдельные копии одинаковых данных. Изменения, внесенные в одну копию, не влияют на другую.

```
//Определение структуры Man
internal struct Man
{
    public int Age { get; set; }

    public Man(int age) => Age = age;

    public override string ToString() => $"Age of man = {Age}";

}
//Создание 2 объектов структуры
Man me = new Man(age : 20);
Man you = new Man(age: 21);

//Output:  Age of man = 20
Console.WriteLine(me);
//Output:  Age of man = 21
Console.WriteLine(you);

me = you;
you.Age = 25;

//Output:  Age of man = 25
Console.WriteLine(you);
//Output:  Age of v = 21
Console.WriteLine(me);
```

Типы записей могут быть либо ссылочными типами (`record class`), либо типами значений (`record struct`).

Как правило, классы используются для моделирования более сложного поведения. Классы обычно хранят данные, которые должны быть изменены после создания объекта класса. Структуры лучше всего подходят для небольших структур данных. Структуры обычно хранят данные, которые не

должны изменяться после создания структуры. Типы записей – это структуры данных с дополнительными членами, синтезированными компилятором. Записи обычно хранят данные, которые не должны изменяться после создания объекта.

```
//Определение записи MyRecord
internal record MyRecord (int A, int B){
    public override string ToString() => $"{A + B}";

}
//Создание объекта MyRecord
MyRecord myRecord = new(10, 20);
//Output: 30
Console.WriteLine(myRecord);
```

Члены типа

К членам типа относятся все методы, поля, константы, свойства и события. В C# нет глобальных переменных или методов, как в некоторых других языках. Даже точка входа программы, метод, должна быть объявлена в классе или структуре (или неявно при использовании операторов верхнего уровня).

Список различных типов членов, которые могут быть объявлены в классе, структуре или записи:

- Поля
- Константы
- Свойства
- Методы
- Конструкторы
- События
- Финализаторы/деструкторы
- Индексаторы
- Операторы
- Вложенные типы

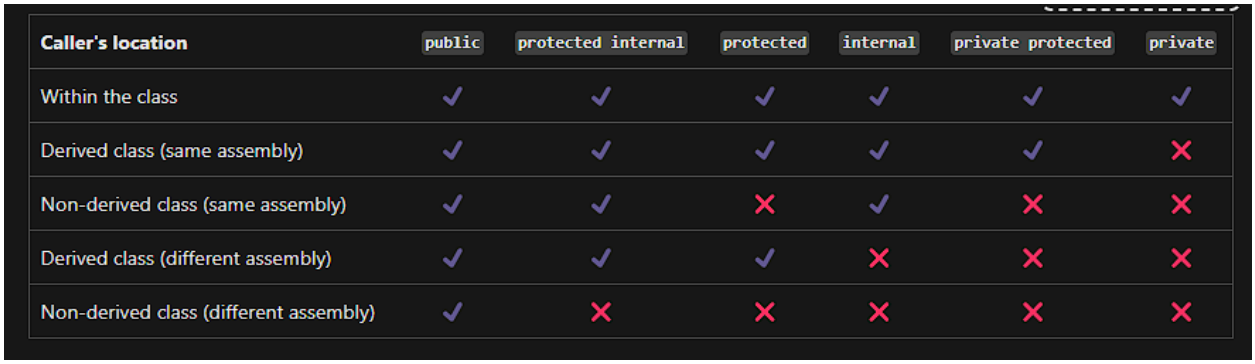
Модификаторы доступа

Некоторые методы и свойства предназначены для вызова или доступа к ним из кода, находящегося вне класса или структуры, называемого клиентским кодом. Другие методы и свойства могут использоваться только в самом классе или структуре. Важно ограничить доступность кода, чтобы к нему мог получить доступ только предполагаемый клиентский код. Вы

указываете, насколько типы и их члены доступны клиентскому коду, с помощью следующих модификаторов доступа:

- public;
- protected;
- internal;
- protected internal;
- private;
- private protected.

По умолчанию модификатор доступа – private



Caller's location	public	protected internal	protected	internal	private protected	private
Within the class	✓	✓	✓	✓	✓	✓
Derived class (same assembly)	✓	✓	✓	✓	✓	✗
Non-derived class (same assembly)	✓	✓	✗	✓	✗	✗
Derived class (different assembly)	✓	✓	✓	✗	✗	✗
Non-derived class (different assembly)	✓	✗	✗	✗	✗	✗

Рисунок 2 – Модификаторы доступа

На рисунке 2 показаны модификаторы доступа.

Конструкторы

Всякий раз, когда создается экземпляр класса или структуры, вызывается его конструктор. Класс или структура может иметь несколько конструкторов, принимающих различные аргументы. Конструкторы позволяют программисту задавать значения по умолчанию, ограничивать число установок и писать код, который является гибким и удобным для чтения.

Существует несколько действий, которые являются частью инициализации нового экземпляра. Эти действия выполняются в следующем порядке:

1. *Поля экземпляра имеют значение 0*. Обычно это делает среда выполнения.

2. *Выполняются инициализаторы полей*. Инициализаторы полей в наиболее производном типе выполняются.

3. *Выполняются инициализаторы полей базового типа*. Инициализаторы полей, начиная с прямого базового типа до каждого базового типа `System.Object`

4. *Выполняются конструкторы базовых экземпляров*. Любые конструкторы экземпляров, начиная с `Object.Object` каждого базового класса до прямого базового класса.

5. *Выполняется конструктор экземпляра*. Конструктор экземпляра для выполнения типа.

6. *Выполняются инициализаторы объектов*. Если выражение содержит инициализаторы объектов, они выполняются после выполнения конструктора экземпляра. Инициализаторы объектов выполняются в текстовом порядке.

Предыдущие действия выполняются при инициализации нового экземпляра. Если для нового экземпляра задано значение *`default`*, для всех полей экземпляра *`struct`* устанавливается значение 0.

```
public class Person
{
   private string last;
   private string first;

   public Person(string lastName, string firstName)
   {
      last = lastName;
      first = firstName;
   }

   // Remaining implementation of Person class.
}
```

Enums

Тип перечисления (или тип *`enum`*) –это тип значения, определенный набором именованных констант применяемого целочисленного типа. Чтобы определить тип перечисления, используйте ключевое слово `enum` и укажите имена элементов перечисления:

```
enum Season
{
    Spring,
    Summer,
    Autumn,
    Winter
}
```

По умолчанию связанные значения констант элементов перечисления имеют тип *`int`*. Они начинаются с нуля и увеличиваются на единицу в соответствии с порядком текста определения. Вы можете явно указать любой

другой целочисленный тип в качестве базового типа перечисления. Вы можете также явно указать соответствующие значения констант, как показано в следующем примере:

```
enum ErrorCode : ushort
{
    None = 0,
    Unknown = 1,
    ConnectionLost = 100,
    OutlierReading = 200
}
```

Вы не можете определить метод внутри определения типа перечисления. Чтобы добавить функциональные возможности к типу перечисления, создайте метод расширения.

Значением по умолчанию для типа перечисления E является значение, созданное выражением `(E)0`, даже если ноль не имеет соответствующего элемента перечисления.

Тип перечисления используется для представления выбора из набора взаимоисключающих значений или комбинации вариантов выбора. Чтобы представить комбинацию вариантов выбора, определите тип перечисления как битовые флаги.

Индивидуальные задания для лабораторной работы

– Согласно варианта описать объект с помощью класса. В нем должно быть не менее трех полей(свойств), желательно, разных типов.

- Создать класс, который будет реализовывать:
    1. Метод для создания массива объектов.
    2. Метод для заполнения массива объектами.
    3. Метод для вывода массива объектов на экран.
    4. Метод для удаления экземпляра класса из массива.
    5. Метод для добавления экземпляра класса в массив.
- Реализовать консольное меню, для проверки функционала.
- В качестве типа одного из полей класса использовать enum (например, для цвета).

Варианты:

1. «Человек».
2. «Учащийся».
3. «Работник».

4. «Транспортное средство».
5. «Инструмент».
6. «Продукт».
7. «Дом».
8. «Книга».
9. «Телевизор».
10. «Компьютер».
11. «Фигура».
12. «Завод».
13. «Животное».
14. «Растение».
15. «Гриб».
16. «Вирус».

Пример выполнения:

```
namespace DevelopmentSoftware;

//Объявление и определение класса предметной области
public class MyClass
{
    //Определение свойств класса
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public MyEnum MyEnum { get; set; }
    //Конструкторы класса
    public MyClass() { }
    public MyClass(int id, string name, string description) =>
(Id, Name, Description) = (id,name,description);
    //Методы класса
    public override string ToString() => $" Id = {this.Id} Name
= {this.Name} description = {this.Description}";

}
//Объявление и определение класса контейнера
public class MyClassContainer
{
    //Объявление массива для хранения объектов класса
    public MyClass[] myClasses { get; set; }
    //Конструкторы класса
    public MyClassContainer() {
        myClasses = new MyClass[0];
    }
```

```
    public MyClassContainer(int size)
    {
        myClasses = new MyClass[size];
    }

    public MyClassContainer(MyClass[] myClasses)
    {
        this.myClasses = myClasses;
    }
    //Методы класса
    public void SortArray()
    {
        //TODO
    }
    public void PrintMyClasses()
    {
        foreach(var elem in myClasses)
        {
            Console.WriteLine(elem);
        }
    }
}
// Пример enum
public enum MyEnum
{
    NO = 0,
    YES = 1,
}
```

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть лабораторной работы.

2. Реализовать индивидуальное задание согласно варианту, сделать скриншоты работающих программ. Написать комментарии.

3. Написать отчет, содержащий:

1. Титульный лист, на котором указывается:

а) полное наименование министерства образование и название учебного заведения;

б) название дисциплины;

в) номер практического занятия;

г) фамилия преподавателя, ведущего занятие;

д) фамилия, имя и номер группы студента;

е) год выполнения лабораторной работы.

2. Индивидуальное задание (листинг кода программы, скриншоты консоли с результатами работы).

3. Вывод о проделанной работе.