Structure

Структура (классическая) -это конструкция языка, позволяющая содержать в себе набор полей различных типов

```
struct Struct
{
    int field;
    static int second;
}
```

В структурах нельзя инициализировать поля непосредственно в месте создания. Инициализация статических полей не обязательна

Struct

Структура в C# - это конструкция языка, состоящая из ключевого слова `struct`, идентификатора и тела. Структуры могут включать в свое тело другие структуры и классы, но такой подход не является широко распространённой техникой

Структуры хранятся в стеке и обычно используются для инкапсуляции небольших групп связанных переменных

Структуры должны использоваться только для хранения маленьких, единых, желательно неизменных значений, которые не будут часто упаковываться

Используйте структуры для упрощения списков параметров методов

Структуры могут содержать **конструкторы**, **константы, поля, методы, свойства, индексаторы, операторы, события и вложенные типы**, однако, если требуются несколько таких членов, рекомендуется и спользовать тип `class`

Стек – это специальная область памяти в которой хранятся структурные типы.

Отличия классов от структур

1. Размещение: в области стека (классы — управляемая куча (heap))

2. Копирование: создаётся отдельная копия объекта, которая после копирования живёт «своей жизнью» (классы — создаётся ссылка на тот же класс (т. н. instance))

3. Наследование: не разрешается дополнение своими свойствами; от него нельзя наследовать (класс — позволяет, кроме случаев когда класс создавался с ключевым словом `sealed`, не разрешающим наследование)

4. Передача параметров: как локальные копии переменных (в классах — как ссылки)

5. Конструктор: да, кроме конструктора по умолчанию, который не требует параметров (в классах — да, без ограничений)

6. Освобождение переменной: при выходе за пределы видимости (в классах — во время процесса сборки мусора (garbage collector))

Конструкторы

При наличии пользовательского конструктора, все неинициализированные поля должны быть в нем инициализированы

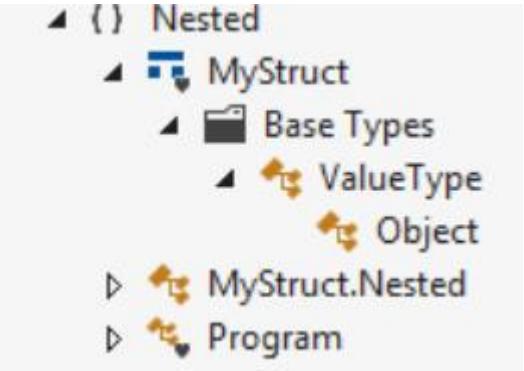
```
struct Struct
{
    int field;
    int item;

     public Struct(int item)
     {
          this.item=item;
          field = 4;
     }
}
```

Структурам запрещается иметь конструктор по умолчанию

Наследование

Структуры могут реализовывать интерфейсы, но они не могут наследоваться от структурили классов, по сколько уже наследуются не явно от абстрактного класса ValueType



В C# можно наследоваться только от одного класса и множество интерфейсов. **От структур наследоваться запрещено**

Nested structand class

Структуры могут содержать вложенные структуры и классы

```
struct MyStruct
{
    public struct Nested
    {
        public void Method()
        {
        }
    }
}
```

```
struct MyStruct
{
    public class Nested
    {
        public void Method()
        {
        }
    }
}
```

Поведение структур

- Структуры размещаются в стеке .
- При копировании структур, создаётся отдельная копия объекта, которая живёт «своей жизнью».
- От структур нельзя наследоваться.
- Структуры могут наследоваться только от интерфейсов.
- Не разрешается дополнение своими свойствами;
- При передаче структуры как параметра передается вся структура.
- В структуре нельзя создать конструктор по умолчанию.
- Структура уничтожается при выходе за пределы видимости.

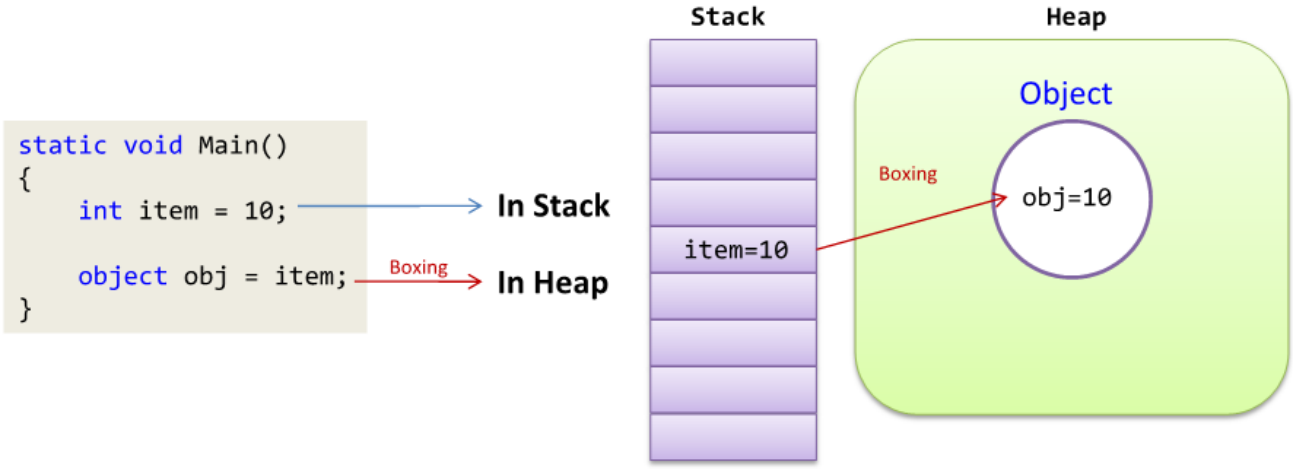
Boxing/UnBoxing

Упаковка представляет собой процесс преобразования структурного типа в тип `object` или любой другой тип интерфейса, реализуемый этим типом**.** Операция распаковки извлекает структурный тип из объекта. Когда структурный тип упаковывается средой CLR, она создает программу-оболочку

значения внутри System.Object и сохраняет ее в управляемой куче

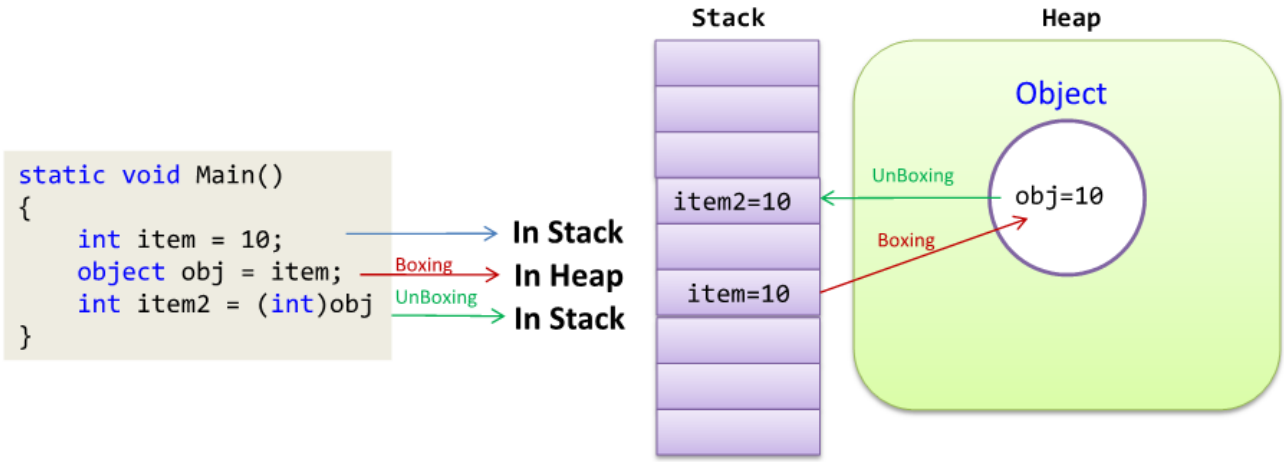
Boxing

Упаковка - преобразование является неявным



UnBoxing

Распаковка-преобразование является явным



Array Covariant

Ковариантность неприменима к массивам элементов структурных типов

```
Dog[] dogs = { new Dog(), new Dog(), new Dog() };

//IAnimal[] animal = dogs; // Ковариантность.
//dogs = array;  // Контрвариантность.

 int[] vector = new int[3] { 1, 2, 3 };
 //object[] array = vector; // Ковариантность
```

DateTime

Структура `DateTime` представляет текущее время, обычно выраженное как дата и время суток

Тип значения `DateTime` представляет дату и время в диапазоне от 00:00:00 1 января 0001 года(н.э.) и до 23:59:59 31 декабря 9999 года(н.э.)

`DateTime.Now` - возвращает объект `System.DateTime`, которому присвоены текущие дата и время суток данного компьютера

Значения времени измеряются в 100-наносекундных единицах, называемых **тактами**, и точная дата представляется числом тактов с 00:00 1 января 0001 года н. э. (н. э.) в календаре **GregorianCalendar** (за исключением тактов, добавленных корректировочными секундами). Например, значение тактов, равное 3124137600000000L, представляет пятницу 1 января 0100 года 00:00:00.

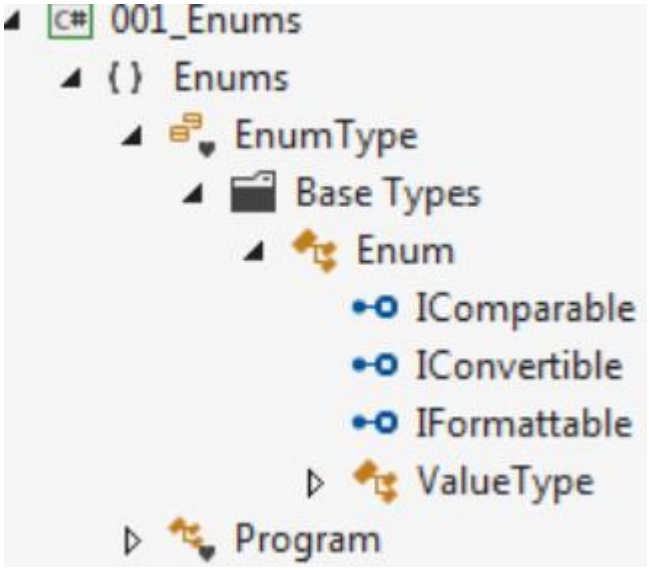
Значение **DateTime** всегда выражается в контексте явно определенного или заданного по умолчанию календаря

Можно создать новое значение DateTime, используя один из следующих способов:

- Путем вызова любой из перегруженных версий конструктора DateTime, которые позволяют указать определенные элементы значения даты и времени (например, год, месяц, день или количество интервалов)
- Используя любой синтаксис компилятора для объявления значений даты и времени.(DateTime.Now)
- Путем анализа строкового представления значения даты и времени. Метод (Parse)

Enum

Перечисление – это конструкция языка которая содержит в себе набор именованных констант, которые хранят в себе определённое значение



Перечисления наследуется от `Enum`, который наследуется от `ValueType` по этому они относятся к категории структурных типов

Рекомендуется называть перечисления существительными в единственном числе

Не рекомендуется создавать перечисления с одним значением

Тип перечисления

Перечислимый тип определяется как набор идентификаторов, с точки зрения языка играющих ту же роль, что и обычные именованные константы, но связанные с этим типом

```
enum EnumType : byte
{
        Zero = 0,
        One = 1,
        Two = 2,
        Three = 3
}
```

Явно указан тип перечисления byte

Использование перечислений позволяет сделать исходные коды программ более читаемыми, так как позволяют заменить «**магические числа**», кодирующие определённые значения, на читаемые имена