

# ОСНОВЫ ТИПОВ

# Все типы — производные от System.Object

- В CLR каждый объект прямо или косвенно является производным от System.Object.
- Это значит, что следующие определения типов идентичны:  
// Тип, неявно производный от
- Object class Employee { ... }
- // Тип, явно производный от
- Object class Employee : System.Object { ... }

**Примитивные**

**Ссылочные**

**Значимые типы**

Примитивный тип	FCL-тип	Совместимость с CLS	Описание
sbyte	System.Sbyte	Нет	8-разрядное значение со знаком
byte	System.Byte	Да	8-разрядное значение без знака
short	System.Int16	Да	16-разрядное значение со знаком
ushort	System.UInt16	Нет	16-разрядное значение без знака
int	System.Int32	Да	32-разрядное значение со знаком
uint	System.UInt32	Нет	32-разрядное значение без знака
long	System.Int64	Да	64-разрядное значение со знаком
ulong	System.UInt64	Нет	64-разрядное значение без знака
char	System.Char	Да	16-разрядный символ Unicode (char никогда не представляет 8-разрядное значение, как в неуправляемом коде на C++)
float	System.Single	Да	32-разрядное значение с плавающей точкой в стандарте IEEE
double	System.Double	Да	64-разрядное значение с плавающей точкой в стандарте IEEE
bool	System.Boolean	Да	Булево значение (true или false)
decimal	System.Decimal	Да	128-разрядное значение с плавающей точкой повышенной точности, часто используемое для финансовых расчетов, где недопустимы ошибки округления. Один разряд числа — это знак, в следующих 96 разрядах помещается само значение, следующие 8 разрядов — степень числа 10, на которое делится 96-разрядное число (может быть в диапазоне от 0 до 28). Остальные разряды не используются

C# разрешает неявное приведение типа, если это преобразование «безопасно», то есть не сопряжено с потерей данных; пример — преобразование из Int32 в Int64.

- Int32 i = 5; // Неявное приведение Int32 к Int32
- Int64 l = i; // Неявное приведение Int32 к Int64
- Single s = i; // Неявное приведение Int32 к Single
- Byte b = (Byte) i; // Явное приведение Int32 к Byte
- Int16 v = (Int16) s; // Явное приведение Single к Int16

- Помимо приведения, компилятор **«знает»** и о другой особенности примитивных типов: к ним применима литеральная форма записи. Литералы сами по себе считаются экземплярами типа, поэтому можно вызывать экземплярные методы, например, следующим образом:

```
Console.WriteLine(123.ToString() + 456.ToString()); // "123456"
```

# Ссылочные типы :

- память для ссылочных типов всегда выделяется из управляемой кучи;
- каждый объект, размещаемый в куче, содержит дополнительные члены, подлежащие инициализации;
- незанятые полезной информацией байты объекта обнуляются (это касается полей);
- размещение объекта в управляемой куче со временем инициирует сборку мусора.

# Значимые типы :

- Экземпляры этих типов обычно размещаются в стеке потока
- В представляющей экземпляр переменной нет указателя на экземпляр;
- Поля экземпляра размещаются в самой переменной
- Поскольку переменная содержит поля экземпляра, то для работы с экземпляром не нужно выполнять разыменование (dereference) экземпляра
- Благодаря тому, что экземпляры значимых типов не обрабатываются уборщиком мусора, уменьшается интенсивность работы с управляемой кучей и сокращается количество сеансов уборки мусора, необходимых приложению на протяжении его существования.

Если тип называют классом (class), речь идет о ссылочном типе. Например, классы `System.Object`, `System.Exception`, `System.IO.FileStream` и `System.Random` — это ссылочные типы

В свою очередь, значимые типы в документации называются структурами (structure) и перечислениями (enumeration). Например, структуры `System.Int32`, `System.Boolean`, `System.Decimal`, `System.TimeSpan` и перечисления `System.DayOfWeek`, `System.IO.FileAttributes` и `System.Drawing.FontStyle` являются значимыми типами

```

// Ссылочный тип (поскольку 'class')
class SomeRef { public Int32 x; }

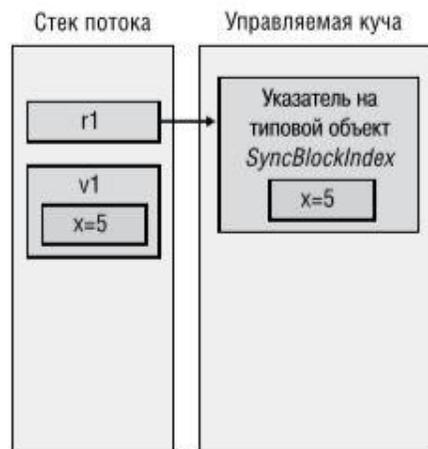
// Значимый тип (поскольку 'struct')
struct SomeVal { public Int32 x; }

static void ValueTypeDemo() {
    SomeRef r1 = new SomeRef(); // Размещается в куче
    SomeVal v1 = new SomeVal(); // Размещается в стеке
    r1.x = 5; // Разыменовывание указателя
    v1.x = 5; // Изменение в стеке
    Console.WriteLine(r1.x); // Отображается "5"
    Console.WriteLine(v1.x); // Также отображается "5"
    // В левой части рис. 5.2 показан результат
    // выполнения предыдущих строк

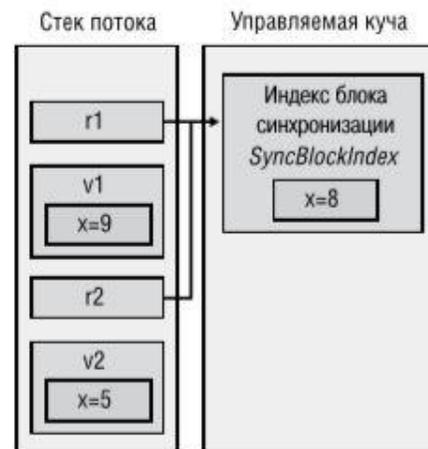
    SomeRef r2 = r1; // Копируется только ссылка (указатель)
    SomeVal v2 = v1; // Помещаем в стек и копируем члены
    r1.x = 8; // Изменяются r1.x и r2.x
    v1.x = 9; // Изменяется v1.x, но не v2.x
    Console.WriteLine(r1.x); // Отображается "8"
    Console.WriteLine(r2.x); // Отображается "8"
    Console.WriteLine(v1.x); // Отображается "9"
    Console.WriteLine(v2.x); // Отображается "5"
    // В правой части рис. 5.2 показан результат
    // выполнения ВСЕХ предыдущих строк
}

```

Состояние после выполнения первой половины метода *ValueTypeDemo*



Состояние после окончательного выполнения метода *ValueTypeDemo*



```
SomeVal v1 = new SomeVal(); // Размещается в стеке
```

Может показаться, что экземпляр `SomeVal` будет помещен в управляемую кучу. Однако поскольку компилятор C# «знает», что `SomeVal` является значимым типом, в сгенерированном им коде экземпляр `SomeVal` будет помещен в стек потока

```
SomeVal v1; // Размещается в стеке
```

Здесь тоже создается IL-код, который помещает экземпляр `SomeVal` в стек потока и обнуляет все его поля. Единственное отличие в том, что экземпляр, созданный оператором `new`, C# «считает» инициализированным.

- // Две следующие строки компилируются, так как C# считает,  
// что поля в v1 инициализируются нулем

```
SomeVal v1 = new SomeVal();
```

```
Int32 a = v1.x;
```

```
// Следующие строки вызовут ошибку компиляции, поскольку C# не считает,  
// что поля в v1 инициализируются нулем
```

```
SomeVal v1;
```

```
Int32 a = v1.x;
```

```
// error CS0170: Use of possibly unassigned field 'x'
```

```
// (ошибка CS0170: Используется поле 'x', которому не присвоено значение)
```

# Важнейшие отличия между значимыми и ссылочными типами:

- Объекты значимого типа существуют в двух формах (см. следующий раздел): неупакованной (unboxed) и упакованной (boxed). Ссылочные типы бывают только в упакованной форме.
- Значимые типы являются производными от `System.ValueType`. Этот тип имеет те же методы, что и `System.Object`. Однако `System.ValueType` переопределяет метод `Equals`, который возвращает `true`, если значения полей в обоих объектах совпадают. Кроме того, в `System.ValueType` переопределен метод `GetHashCode`, который создает хеш-код по алгоритму, учитывающему значения полей экземпляра объекта. Из-за проблем с производительностью в реализации по умолчанию, определяя собственные значимые типы значений, надо переопределить и написать свою реализацию методов `Equals` и `GetHashCode`
- Поскольку в объявлении нового значимого или ссылочного типа нельзя указывать значимый тип в качестве базового класса, создавать в значимом типе новые виртуальные методы нельзя. Методы не могут быть абстрактными и неявно являются запечатанными (то есть их нельзя переопределить).

- Переменные ссылочного типа содержат адреса объектов в куче. Когда переменная ссылочного типа создается, ей по умолчанию присваивается null, то есть в этот момент она не указывает на действительный объект. Попытка задействовать переменную с таким значением приведет к генерации исключения `NullReferenceException`. В то же время в переменной значимого типа всегда содержится некое значение соответствующего типа, а при инициализации всем членам этого типа присваивается 0. Поскольку переменная значимого типа не является указателем, при обращении к значимому типу исключение `NullReferenceException` возникнуть не может. CLR поддерживает понятие значимого типа особого вида, допускающего присваивание null
- Когда переменной значимого типа присваивается другая переменная значимого типа, выполняется копирование всех ее полей. Когда переменной ссылочного типа присваивается переменная ссылочного типа, копируется только ее адрес.
- Вследствие сказанного в предыдущем пункте несколько переменных ссылочного типа могут ссылаться на один объект в куче, благодаря чему, работая с одной переменной, можно изменить объект, на который ссылается другая переменная. В то же время каждая переменная значимого типа имеет собственную копию данных «объекта», поэтому операции с одной переменной значимого типа не влияют на другую переменную.
- Так как неупакованные значимые типы не размещаются в куче, отведенная для них память освобождается сразу при возвращении управления методом, в котором описан экземпляр этого типа (в отличие от ожидания уборки мусора).