СИСТЕМА ТИПОВ

Единая система типов C# (и .NET)

- Common Type System (CTS) – спецификации Microsoft, описывающие определение типов и их поведение
- Common Language Specification (CLS) – подмножество типов CTS, которые могут быть использованы в коде с разными языками программирования

Единая система типов является основой межъязыкового взаимодействия

- ✔ Единая система типов для C#, Visual Basic, JScript, Pascal, J#, C++
- ✔ Единая библиотека базовых классов
- ✔ Межъязыковое наследование
- ✔ Межъязыковая обработка исключений
- ✔ Переход при отладке между модулями на разных языках

Типы-значения (value types) (размерные, структурные)

- Простые типы-значения (int, float,…)
- Перечисления (enum)
- Структуры (struct)

Ссылочные типы (reference types)

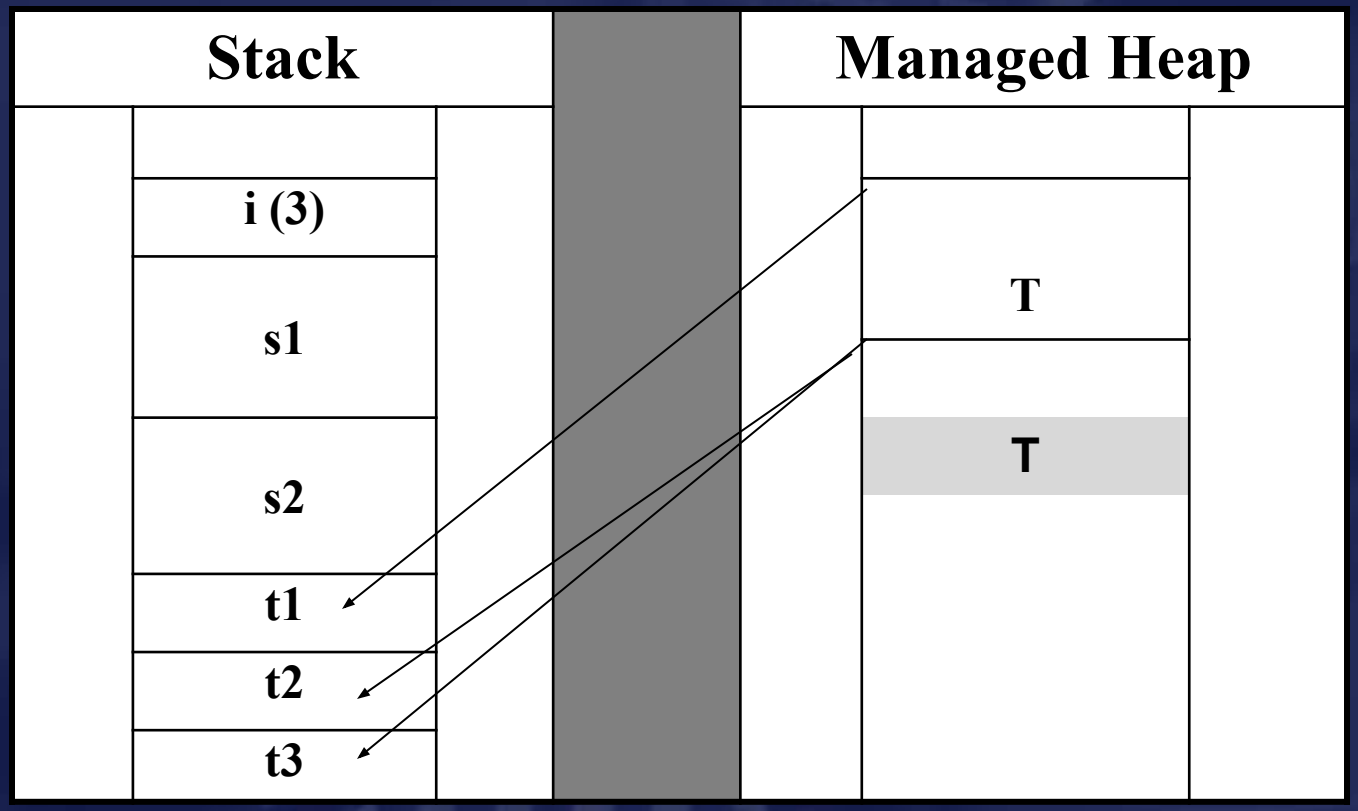
- Классы (class)
- Делегаты (delegate)
- Интерфейсы (interface)
- Строки (System.String)
- System.Object
- Массивы (System.Array)

Ссылочные типы и типы-значения

	Value Types	Reference Types
Переменная содержит…	…значение	…ссылку на значение
Значение хранится в …	… стеке	…динамической памяти (managed heap)
Инициализируется	0, false, ‘\0’	null
При присваивании…	…копируется значение	…копируется ссылка

Ссылочные типы и типы-значения

```
struct S {…};          class T {…};
int i = 3;             T t1 = new T();
S s1 = new S();        T t2 = new T();
S s2 = s1;             T t3 = t2;
```



Встроенные типы-значения

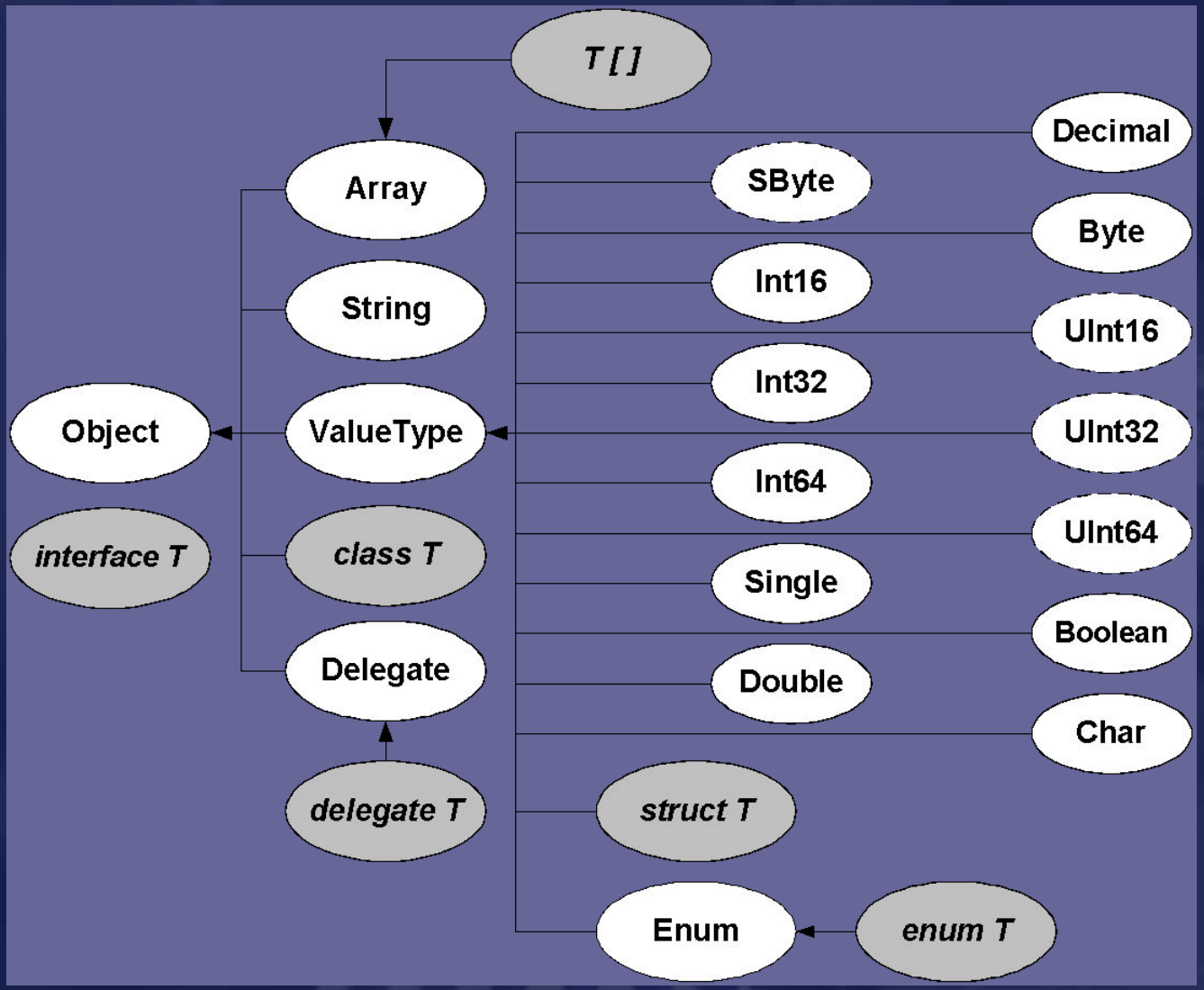
C#	CLS	Тип CTS	Длина
sbyte	нет	System.Sbyte	8 бит
byte	+	System.Byte	8 бит
short	+	System.Int16	16 бит
ushort	нет	System.UInt16	16 бит
int	+	System.Int32	32 бит
uint	нет	System.UInt32	32 бит
long	+	System.Int64	64 бит
ulong	нет	System.UInt64	64 бит
char	+	System.Char	16 бит
float	+	System.Single	32 бит
double	+	System.Double	64 бит
bool	+	System.Boolean	8? бит
decimal	+	System.Decimal	128 бит

Класс Object - самый базовый класс

```
class  Object
{…
public virtual string ToString();
public virtual bool Equals(object obj);
public static bool Equals( object objA, object objB );
public static bool ReferenceEquals( object objA, object objB );

public virtual int GetHashCode();
public Type GetType();
…
}
```

- ✔ Методы класса object могут быть вызваны для объектов любого типа
- ✔ Переменной типа object может быть присвоено значение любого типа

Система типов CLR



Упаковка(boxing) и распаковка(unboxing) -1

- ✔ Упаковка (boxing) - преобразование размерного типа (value type) в ссылочный.
- ✔ Упаковка обычно выполняется при передаче объекта размерного типа (value type) как значение для параметра, имеющего тип object.

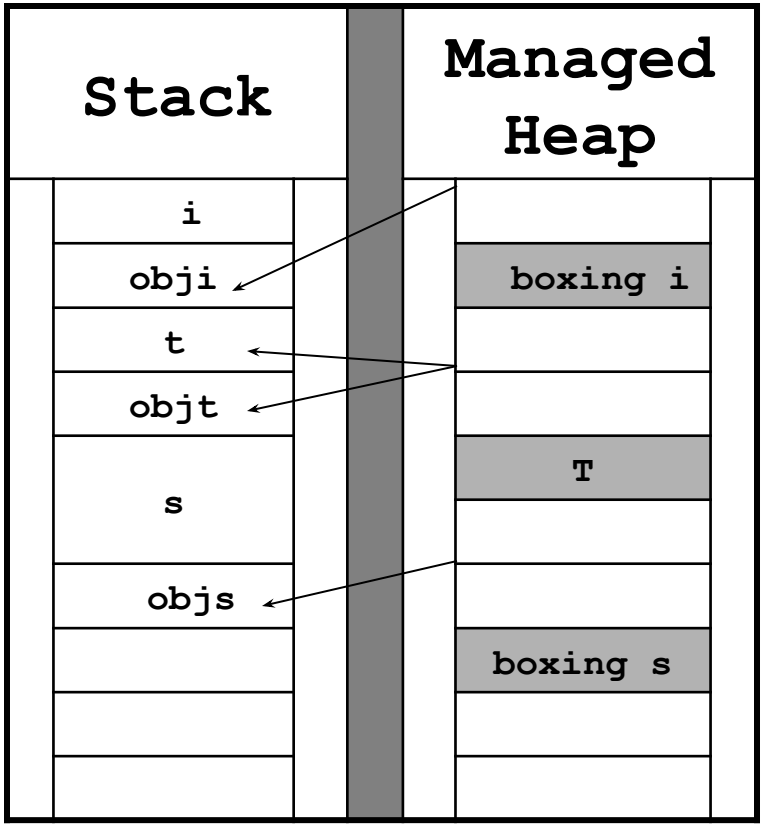
```
int x = 5;
object obj = x;       // Явная упаковка
string s = x.ToString();// Неявная упаковка
s = (123.56).ToString();
int res = (int)obj;     // Распаковка
```

Упаковка(boxing) и распаковка(unboxing) -2

- ✔ При упаковке
  - в управляемой куче выделяется память для объекта;
  - поля объекта копируются в выделенную память в управляемой куче.
- ✔ Распаковка состоит в получении указателя на поля данных исходного размерного типа в управляемой куче.
- ✔ При распаковке копирование полей не выполняется.

Упаковка и распаковка. Пример

```
public class T
{    public int x;
     public T(int x)    { this.x = x; }
     public void SetValue ( int val ) { x = val;}
}
public struct S
{    public int x;
     public S(int x)    { this.x = x; }
     public void SetValue ( int val ) { x = val;}
}
public class Class1{
static void Main(string[] args)
{    int i = 1;
     object obji = i;
     obji = 5;
     Console.WriteLine( " i = {0} obji = {1}", i, obji);
                  // Output i = 1 obji = 5
     T t = new T(1);
     object objt = t;
     t.SetValue(2);
     ((T)objt).SetValue(5);
     Console.WriteLine( " t.x = {0} ((T)objt).x = {1}", t.x, ((T)objt).x);
                       // Output t.x = 5 ((T)objt).x = 5
     S s = new S(1);
     object objs = s;
     s.SetValue(2);
     ((S)objs).SetValue(5);
     Console.WriteLine( " s.x = {0} ((S)objs).x = {1}", s.x, ((S)objs).x);
                       // Output s.x = 2 ((S)objs).x = 1
}}
```



Арифметические типы

- ✔ Неявные преобразования арифметических типов разрешены, если это не приводит к потере информации

```
int iv = 10;
long lv = iv;
```

- ✔ Явное преобразование может привести к потере информации

```
long lv = long.MaxValue;
int iv = (int)lv;
```

Операторы checked и unchecked

- ✔ Только для целочисленных типов проверяется переполнение при выполнении операций

```
try
{   int i0 = int.MaxValue;
    int i1 = i0 + 100;
    int i2 = checked(i0 + 100);
}
catch(OverflowException ex)
{    Console.WriteLine("Try1:\n" + ex.Message );
}
```

Настройка компилятора:

Project / Properties…

Build / Advanced…

Check For Arithmetic Overflow / Underflow (true / false)

Вычисления с плавающей запятой

```
double d1 = 0;
double d2 = 0;
double res = d1 / d2;
Console.WriteLine("d1 = {0} d2 = {1} res = {2}", d1, d2, res);

double d0 = 0;
d1 = -1.0;
d2 = 1.0;

double res1 = d1 / d0;
Console.WriteLine("d1 = {0} d0 = {1} res1 = {2}", d1, d0, res1);
double res2 = d2 / d0;
Console.WriteLine("d2 = {0} d0 = {1} res2 = {2}", d2, d0, res2);

res = res1 + res2;
Console.WriteLine("res1 = {0} res2 = {1} res = {2}", res1, res2, res);
```

```
double d3 = double.PositiveInfinity;
res1 = d3 + 1.23;
res2 = d3 * 0;
Console.WriteLine("d3 = {0} res1 = {1} res2 = {2}", d3, res1, res2);

double d4 = double.NaN;
res1 = d4 * 0;
res2 = d4 / double.PositiveInfinity;
Console.WriteLine("d4 = {0} res1 = {1} res2 = {2}", d4, res1, res2);
```

Результат:

```
d1 = 0 d2 = 0 res = NaN
d1 = -1 d0 = 0 res1 = -бесконечность
d2 = 1 d0 = 0 res2 = бесконечность
res1 = -бесконечность res2 = бесконечность res = NaN
d3 = бесконечность res1 = бесконечность res2 = NaN
d4 = NaN res1 = NaN res2 = NaN
```

Статический класс Convert

✔ Содержит методы для преобразования значений одного базового типа данных к другому базовому типу. В частности, в классе определены методы

```
public static int ToInt32( string value );
public static double ToDouble( string value );
public static int ToInt32( double value ); // с округлением
```

✔ Методы бросают исключение, если преобразование невозможно.

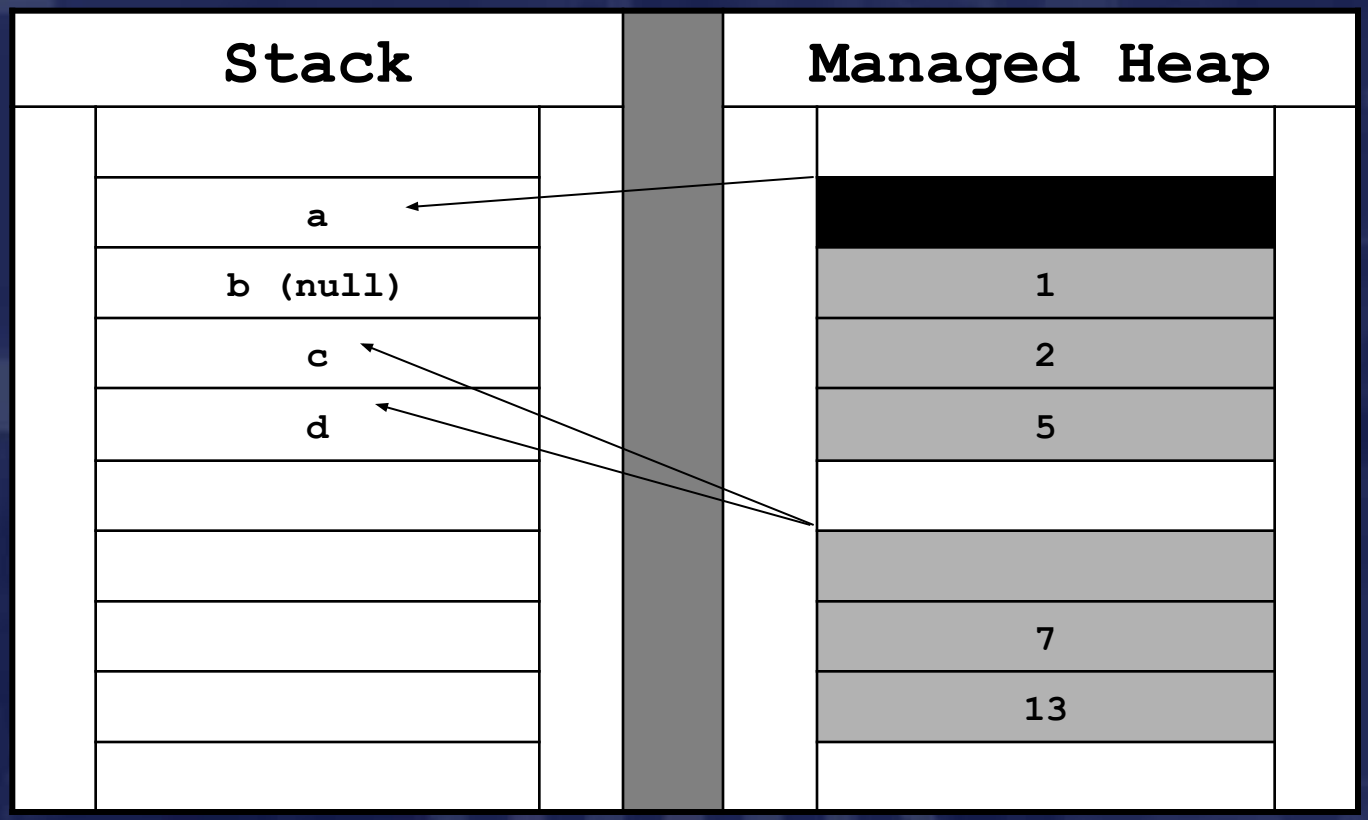
```
try {
  double d1 = 1.5;
  int i1 = Convert.ToInt32(d1);
  Console.WriteLine(i1);          // 2

  double d2 = 2.5;
  int i2 = Convert.ToInt32(d2);
  Console.WriteLine(i2);          // 2

  double d3 = 1.234523452345;
  float f1 = Convert.ToSingle(d3);
  Console.WriteLine(f1);          // 1.234523

  double d4 = double.MaxValue;
  float f2 = Convert.ToSingle(d4);
  Console.WriteLine(f2);              // бесконечность
  int i3 = Convert.ToInt32(d4);       // исключение
  Console.WriteLine(i3);
  }
 catch (Exception ex)
  { Console.WriteLine(ex.Message);  }
```

Массивы

- ✔ Ссылочный тип. Память всегда выделяется в управляемой куче.
- ✔ Абстрактный базовый класс System.Array.
- ✔ CLR поддерживает
  - Одномерные массивы
  - Многомерные массивы
  - Ступенчатые (jagged) массивы (не совместимы с CLS)

Одномерные массивы типов-значений

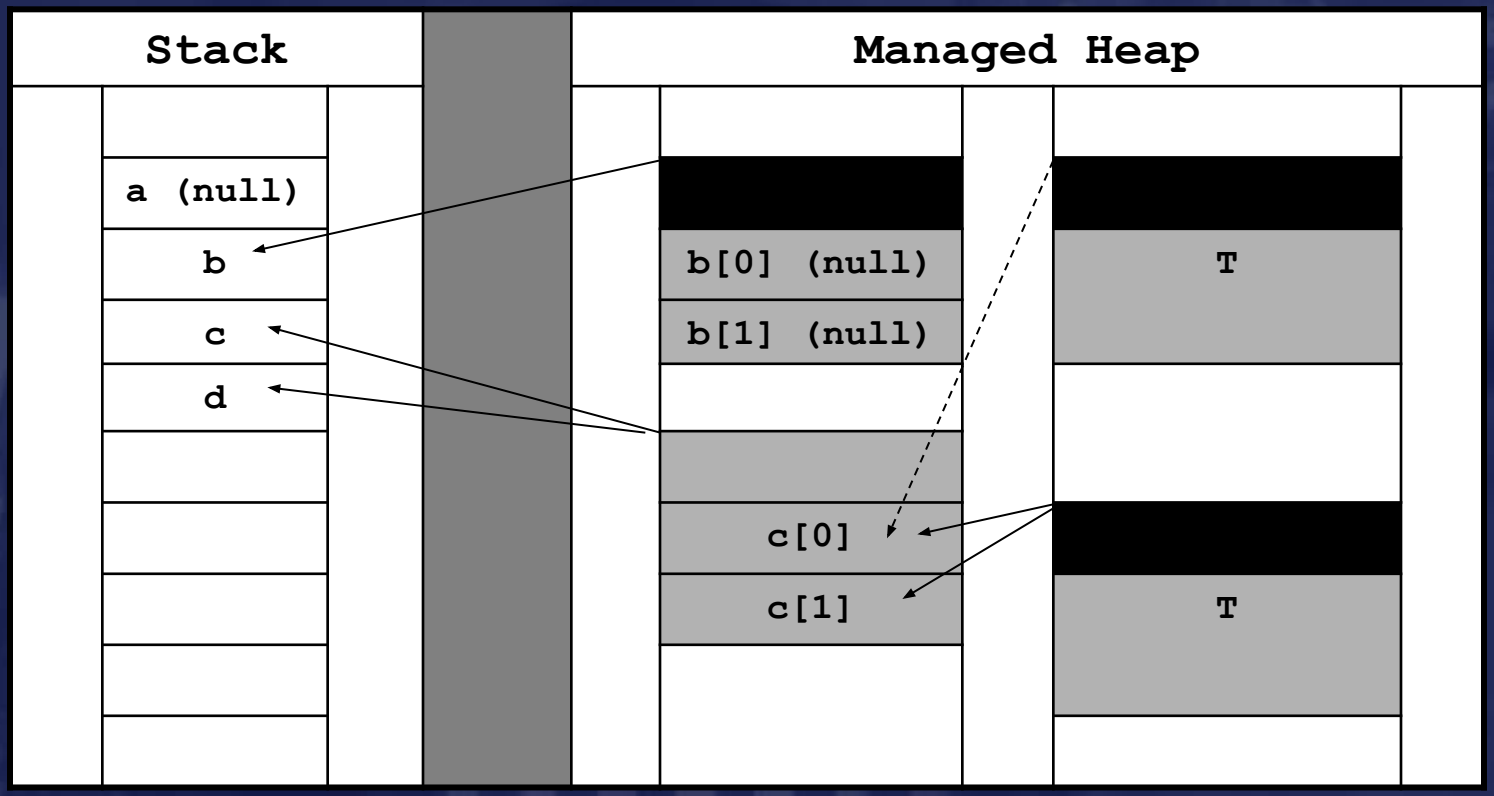
```
int[] a = new int[3] {1,2,5};
int[] b; // b == null
int[] c = { 7,13 };
int[] d = c; // d и c – это один и тот же массив!
```



Одномерные массивы ссылочных типов

```
class T
{ ...
   T(int par1, int par2)
   {...}
 ...
}
```

```
T[] a;
T[] b = new T[2];
T[] c = new T[2] {new T(2,3), new T(12,5)};
c[0] = c[1];
T[] d = c;
```



Выход за границы массива

- ✔ В массивах C# всегда хранится информация о числе измерений массива и числе элементов в каждом измерении.
- ✔ В массивах C# проверяется выход индекса за границы массива.
- ✔ Отключить проверку выхода за границы массива можно только для массивов с элементами размерных типов, включив их в блок unsafe (требует настройки компилятора).

Массивы нулевой длины

- ✔ Можно объявить массив нулевой длины. Массив не содержит элементов, но ссылка отлична от null .

```
 try
   {   double[] arr1 = new double[0];
       Console.WriteLine(arr1.Length);

       double[] arr2 = null;
       Console.WriteLine(arr2.Length);
    }
  catch (Exception ex)
    {  Console.WriteLine(ex.Message);
    }

//  Вывод:
//  0
//  Object reference not set to an instance of an object.
```

Инициализация элементов массива Приведение типов

- По умолчанию при создании массива элементы размерных типов инициализируются нулевыми значениями, элементы ссылочных типов – значением null.
- Неявное преобразование массива типа T1[ ] к массиву типа T2[ ] возможно только, если
  - T1 и T2 – ссылочные типы
  - допустимо неявное преобразование T1->T2
  - массивы имеют одинаковую размерность
- Явное и неявное преобразование массивов с элементами размерных типов (value types) запрещено.

Некоторые методы класса Array

- Свойства для получения размеров массива

```
int[] a = new int[3];

a.Length - число элементов в массиве
a.Rank - число измерений массива
a.GetLength(int i) - размер i-го измерения
```

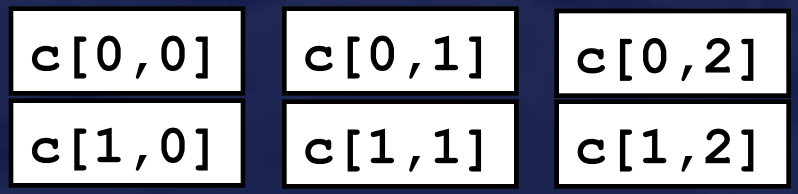
- Копирование одномерного массива

```
int[] a = new int[3];
int[] b = (int[])a.Clone(); // Копирует массив
```

Многомерные массивы

- Прямоугольные массивы

```
int[,] c = new int[2,3] { {1,2,3 }, {4,5,6} };
c[1,2] = 10;
// c.Length == 6
// c.GetLength(0) == 2
// c.GetLength(1) == 3
// c.Rank == 2
```



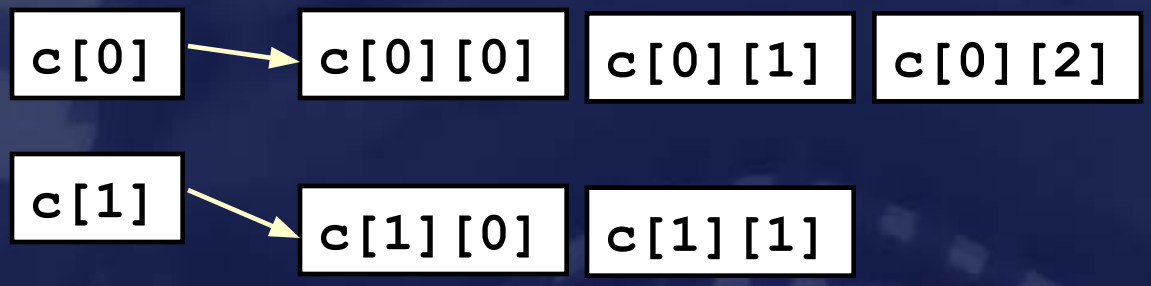
- Можно копировать при помощи Clone().

Многомерные ступенчатые (jagged) массивы

(другие названия - вложенные, зубчатые, невыравненные)

```
int[][] c = new int[2][];
c[0] = new int[3] { 0,1,2 };
c[1] = new int[2] { 3,4 };
…
// c.Length == 2
// c[0].Length == 3
```

- ✔ Разные строки могут иметь разную длину



- ✔ Будьте внимательны при работе с Clone().

Строки. Класс System.String

- ✔ Неизменяемые последовательности символов Unicode.
- ✔ В созданной строке нельзя изменить ни отдельные символы, ни длину. При операциях со строками создаются новые объекты, память для которых распределяется в управляемой куче.
- ✔ При компиляции исходного текста все литеральные строки размещаются в метаданных модуля в одном экземпляре (в хэш-таблице).
- ✔ Нелитеральные строки можно добавить в хэш-таблицу с помощью метода

```
string string.Intern(string);
```

Приемы работы со строками

- ✔ Возможен поэлементный доступ

```
string s = "Hello, World!";
Console.WriteLine(s[0]);       // H
foreach(char c in s) Console.WriteLine("{0}",c);
```

- ✔ Операция сложения определена как конкатенация строк

```
string s1 = "Hello";
string s2 = "World";
string s3 = s1 + "," + s2 + "!" ;
```

Приемы работы со строками -2

- Статический метод Concat (9 перегрузок) выполняет объединение строк или строковых представлений объектов

```
public static string Concat ( params Object[] args ) ;
```

- Статический метод Join (2 перегрузки) выполняет объединение строк, вставляя строку-разделитель между элементами

```
public static string Join ( string separator, string[] value
);
```

- Статический метод Format (5 перегрузок) формирует строку с использованием строки форматирования

```
public static string Format(string format, params
Object[]args);
```

Метод Split

- Метод Split (6 перегрузок) формирует из строки массив строк, используя как разделители заданные символы

```
public string[] Split ( params char[] separator );
```

- Пример

```
string str = "ab cd;abc; 1234";
string[] sar1 = str.Split(';',' ');
foreach (string i in sar1) Console.WriteLine(i);

char[] delim = {';'};
string[] sar2 = str.Split(delim,2);
foreach (string i in sar2) Console.WriteLine(i);
```

Вывод:

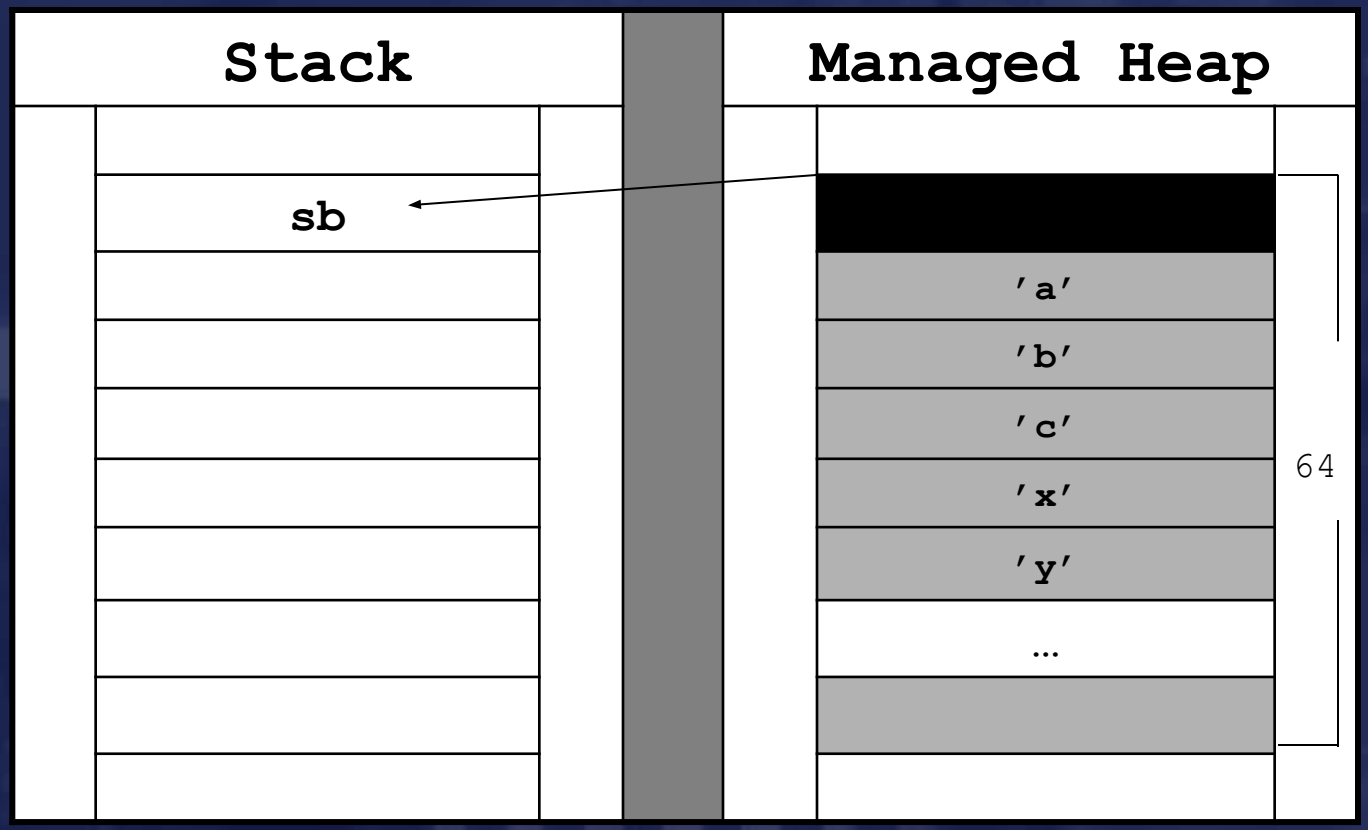
```
ab
cd
abc

1234
ab cd
abc; 1234
```

Строки. Класс System.Text.StringBuilder

- ✔ Изменяемые последовательности символов Unicode.
- ✔ Строки можно модифицировать без перераспределения памяти.
- ✔ При создании объекта (6 Ctors) можно распределить память “с запасом”.
- ✔ Свойство int Capacity { get; set;}.

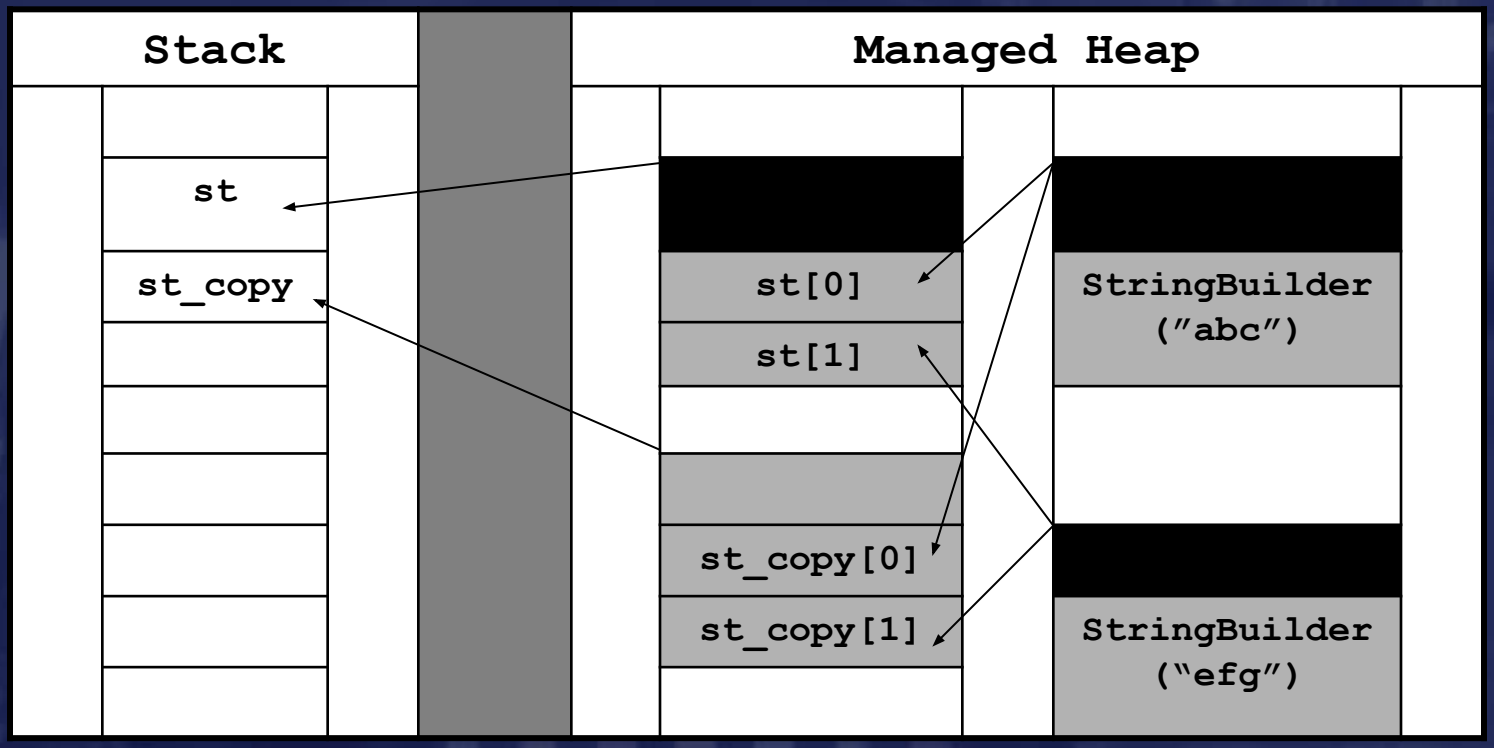
Класс System.Text.StringBuilder - 2

```
StringBuilder sb = new StringBuilder( "abc“, 64);
Console.WriteLine( "{0} {1} {2}",    sb,  sb.Length,  sb.Capacity); //  abc  3  64
sb.Append(“xy”);
Console.WriteLine( "{0} {1} {2}",    sb, sb.Length,  sb.Capacity); //  abcxy  5  64
string str = sb.ToString();
```



Пример Arrays_Demo

```
StringBuilder [] st =     new StringBuilder[2]  {new StringBuilder("abc"), new StringBuilder("efg")};

StringBuilder[] st_copy = (StringBuilder[]) st.Clone();
st[1][2] = 'z';

Console.WriteLine("\nst");
foreach(StringBuilder elem in st) Console.Write("  {0}",elem);                    // abc efz

Console.WriteLine("\nst_copy");
foreach(StringBuilder elem in st_copy) Console.Write("  {0}",elem);              // abc efz
```



Средства консольного ввода/вывода

- Для организации консольного ввода/вывода предназначены статические методы класса System.Console

```
…
Console.WriteLine(“Hello, World!”);
Console.Write(“Hello, World!”);
…
```

- Методы Write и WriteLine определены как методы с переменным числом параметров

```
…
Console.WriteLine(“{0},{1}{2}”,”Hello”,”World”,”!”);
…
```

Консольный ввод

- ✔ Ввод очередного символа и целой строки

```
int i = Console.Read();
string str = Console.ReadLine();
```

- ✔ Преобразование введенной строки в число

```
string str = Console.ReadLine();
int i = Int32.Parse(str);
float f = float.Parse(str);
double d = double.Parse(str);
```

- ✔ При передаче в качестве параметра неправильной строки бросается исключение.

Консольный вывод: форматирование

- Общий вид строки форматирования

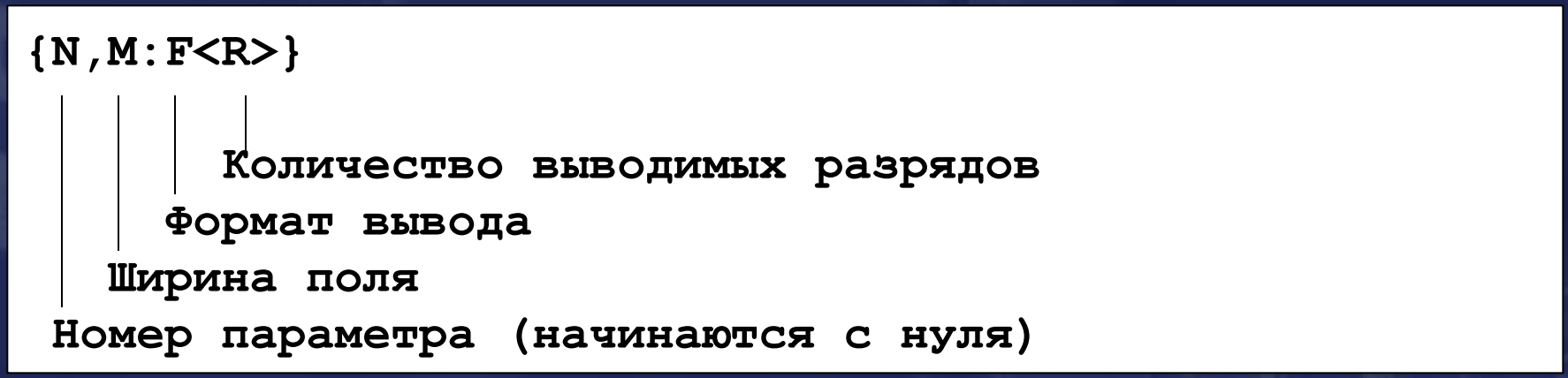
```
{N,M:F<R>}
  |  |  |  |
  |  |  |  Количество выводимых разрядов
  |  |  Формат вывода
  |  Ширина поля
  Номер параметра (начинаются с нуля)
```

- Форматы вывода

```
C - форматирование числа как денежной суммы
D - Целое число
E - Вещественное число в виде 1e+3
F - Вещественное число в виде 123.456
G - Вещественное число в наиболее компактном формате
N - Вещественное число в виде 123,456,789.5
X - Целое число в шестнадцатеричном виде
```

Консольный вывод: форматирование. Пример

Общий вид строки форматирования




```
double d = 1234.56789;
Console.WriteLine(d);
Console.WriteLine("{0,15:E} {1,15:F3}{2,15:G7}",d,d,d);
Console.WriteLine("{0,15:E1} {1,15:F5} {2,15:G9}",d,d,d);
Console.WriteLine("{0,5:E2} {1,5:F3} {2,5:G9}",d,d,d);
// Вывод:
// 1234,56789
//   1,234568E+003       1234,568       1234,568
//         1,2E+003     1234,56789     1234,56789
// 1,23E+003 1234,568 1234,56789
```