Структура программы на C#

- ✔ Любая программа на C# - это набор классов, структур и других определенных программистом типов.
- ✔ Тип (class, struct и interface) можно определить в одном или нескольких файлах. Если тип определяется в нескольких файлах, перед каждым объявлением типа указывается ключевое слово partial. В одном файле может быть определено несколько типов.

```
// файл Book.cs
using System;
namespace First_Sample
{
   partial class Book
   {  private string title;
      private string author;
      private int year;

      public Book(  string title,  string author, int year)
        {  this.title = title;
           this.author = author;
           this.year = year;
         }
      public override string ToString()
        {  return author + " " + title + " " + year;  }
   }
}
```

```
// файл Book1.cs
using System;
namespace First_Sample
{
   partial class Book
   {    public Book (  string title,
                       Person author_data,
                       int year)
          {  this.title = title;
             this.author =  author_data.FirstName + "  "
                          + author_data.SecondName;
             this.year = year;
          }
    }
}
```

Пространства имен. Директива using

- ✔ Пространства имен позволяют избежать совпадения имен в больших проектах.
- ✔ Имя типа, который определен в примере, First_Sample.Person.
- ✔ Директива using дает возможность использовать для типа сокращенное имя.
- ✔ В примере для типа System.DateTime используется сокращенное имя DateTime, так как в начале файла есть директива using System;

```
using System;
using System.Collections.Generic;
namespace First_Sample
{   partial class Person
      {   private string[] names;
          private DateTime date { get;set;}

          public Person(string first_name, string second_name, DateTime date)
            {   names = new string[] { first_name, second_name };
                this.date = date;
            }
   }
}
```

- ✔ Директиву using можно разместить только в начале файла или пространства имен.
- ✔ Пространства имен могут быть вложены. Директива using для объемлющего пространства имен не означает сокращение имен для вложенных using.

Единая система типов .NET

- ✔ В спецификациях Microsoft определена единая система типов платформы .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 для C#, Visual Basic, JScript, Pascal, J#, C++.

Ссылочные типы и типы-значения

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)
- Класс System.Object
- Массивы (System.Array)

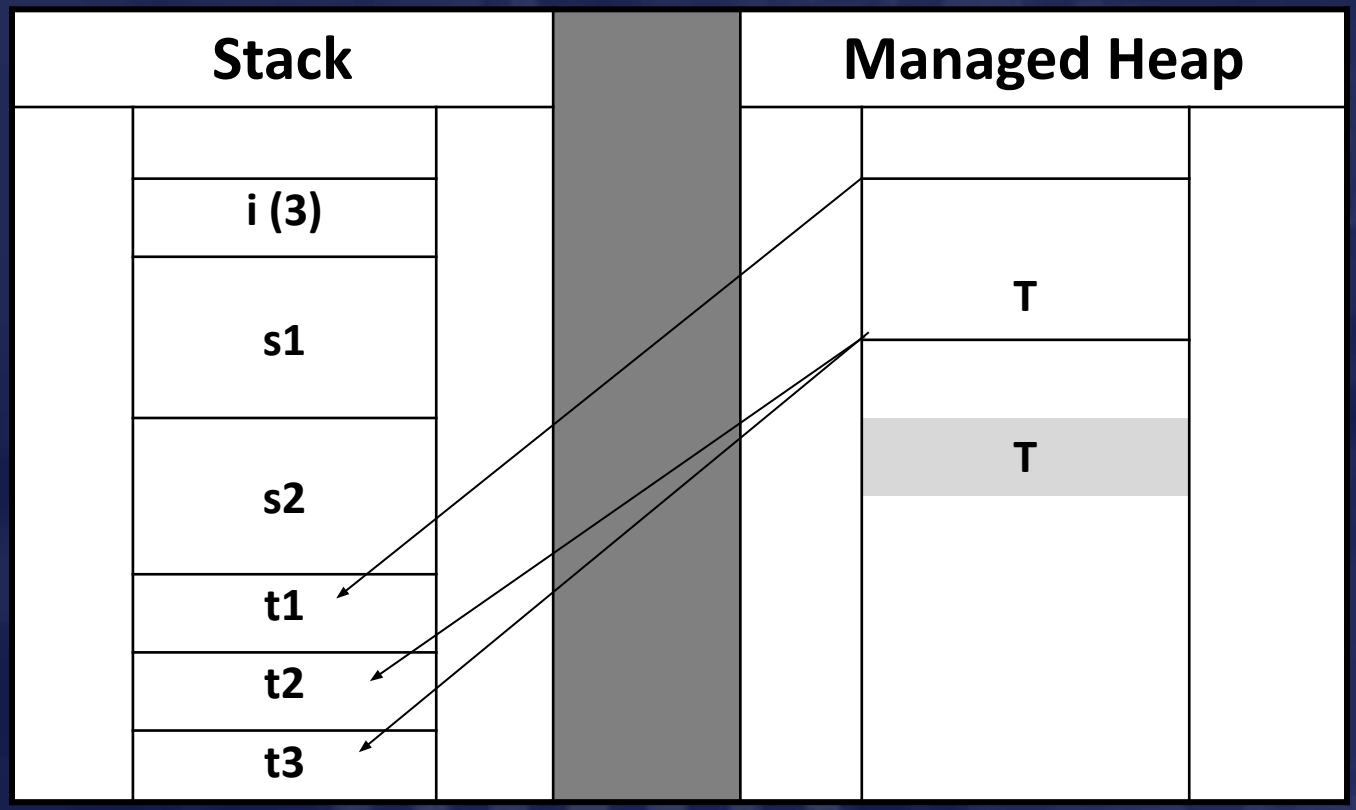
	Value Types	Reference Types
Переменная содержит…	…значение	…ссылку на значение
Значение хранится в …	… стеке	…динамической памяти (managed heap)
Инициализируется	0, false, ‘\0’	null
При присваивании…	…копируется значение	…копируется ссылка

Встроенные типы-значения

C#	Тип CTS	Длина
sbyte	System.Sbyte	8 бит
byte	System.Byte	8 бит
short	System.Int16	16 бит
ushort	System.UInt16	16 бит
int	System.Int32	32 бит
uint	System.UInt32	32 бит
long	System.Int64	64 бит
ulong	System.Int64	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

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;
```



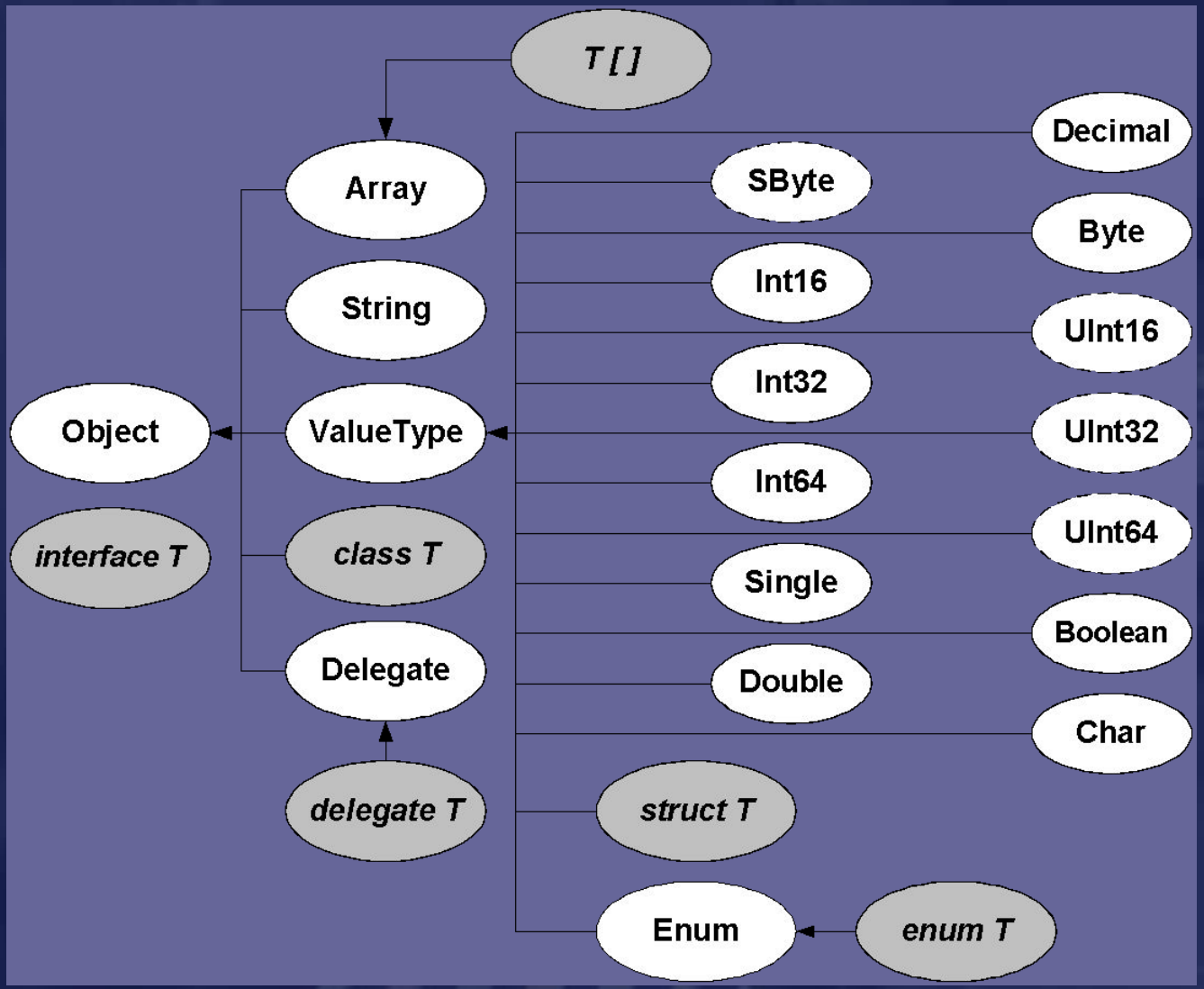
Класс System.Object - самый базовый класс

```
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)

- ✔ 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
```

- ✔ 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

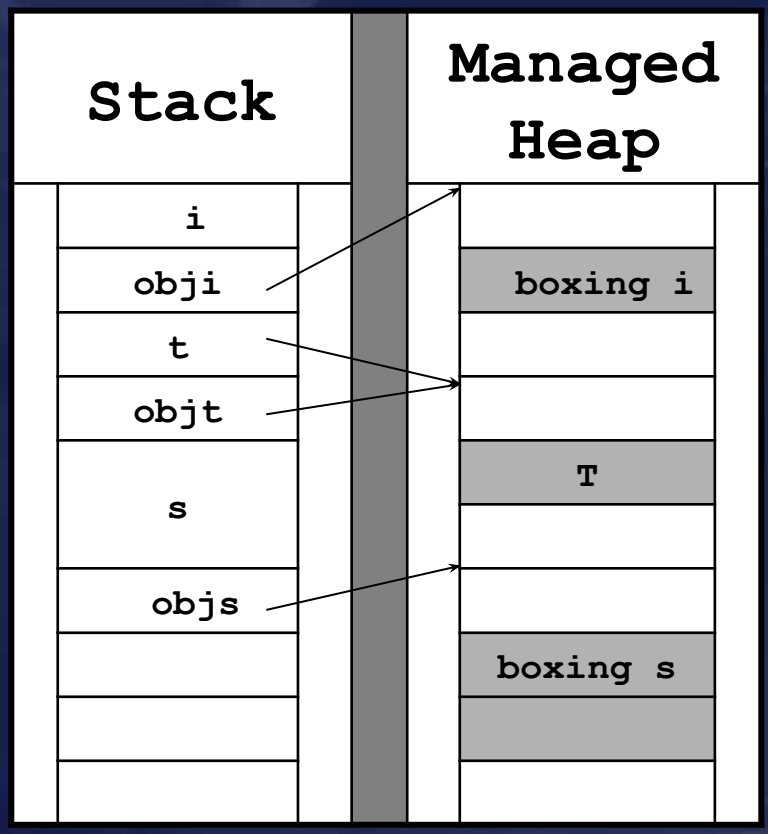
```
public class T
{       public int x;
        public T(int x)     { this.x = x; }
        public void SetValue ( int val ) { x = val;}
}
```

```
public struct S
{       public int x;
        public S(int x)     { this.x = x; }
        public void SetValue ( int val ) { x = val;}
}
```

```
public class Program {
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 long_value = long.MaxValue;
Console.WriteLine("long_value= {0}", long_value);
int int_value = (int)long_value;
Console.WriteLine("int_value= {0}", int_value);
```

Вывод:

```
long_value= 9223372036854775807
int_value= -1
```

- ✔ В C# 7 можно добавлять символ ‘_ ‘ (подчеркивание) как разделитель в числовые литералы. Разделитель можно добавить в любом месте между цифрами, на значение он не влияет.

```
int jval = 1_234_567;
int dval = 1_234.567_89;
```

Операторы checked и unchecked

- ✔ Только для целочисленных типов проверяется переполнение при выполнении арифметических операций

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

Вывод при настройке компилятора:

Check For Arithmetic Overflow / Underflow false

```
i0 = 2147483647
i1 = -2147483549
Переполнение в результате выполнения арифметической операции.
```

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 = -Infinity
d2 = 1 d0 = 0 res2 = Infinity
res1 = -Infinity res2 = Infinity res = NaN
d3 = Infinity res1 = Infinity 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);  }
```

Перечисление (enum)

- ✔ Тип –значение, который состоит из набора именованных констант.

```
enum Duration {Day, Week, Month};
```

- ✔ Каждый тип перечисления имеет базовый тип - любой целочисленный тип кроме char (умолчание int).
- ✔ В примере объявляются переменные типа Duration, которым присваиваются значения и создается массив из всех значений перечисления.

```
Duration duration_1 = Duration.Day;
Duration duration_2 = (Duration)2;
Duration duration_3 = (Duration)10; // можно, плохо

Console.WriteLine("duration_1 = " + duration_1);
Console.WriteLine("duration_2 = " + duration_2);
Console.WriteLine("duration_3 = " + duration_3);

Duration[] array = (Duration[]) Enum.GetValues(typeof(Duration));
for (int j = 0; j < array.Length; j++) Console.WriteLine(array[j]);
```

Вывод:

```
duration_1 = Day
duration_2 = Month
duration_3 = 10
Day
Week
Month
```

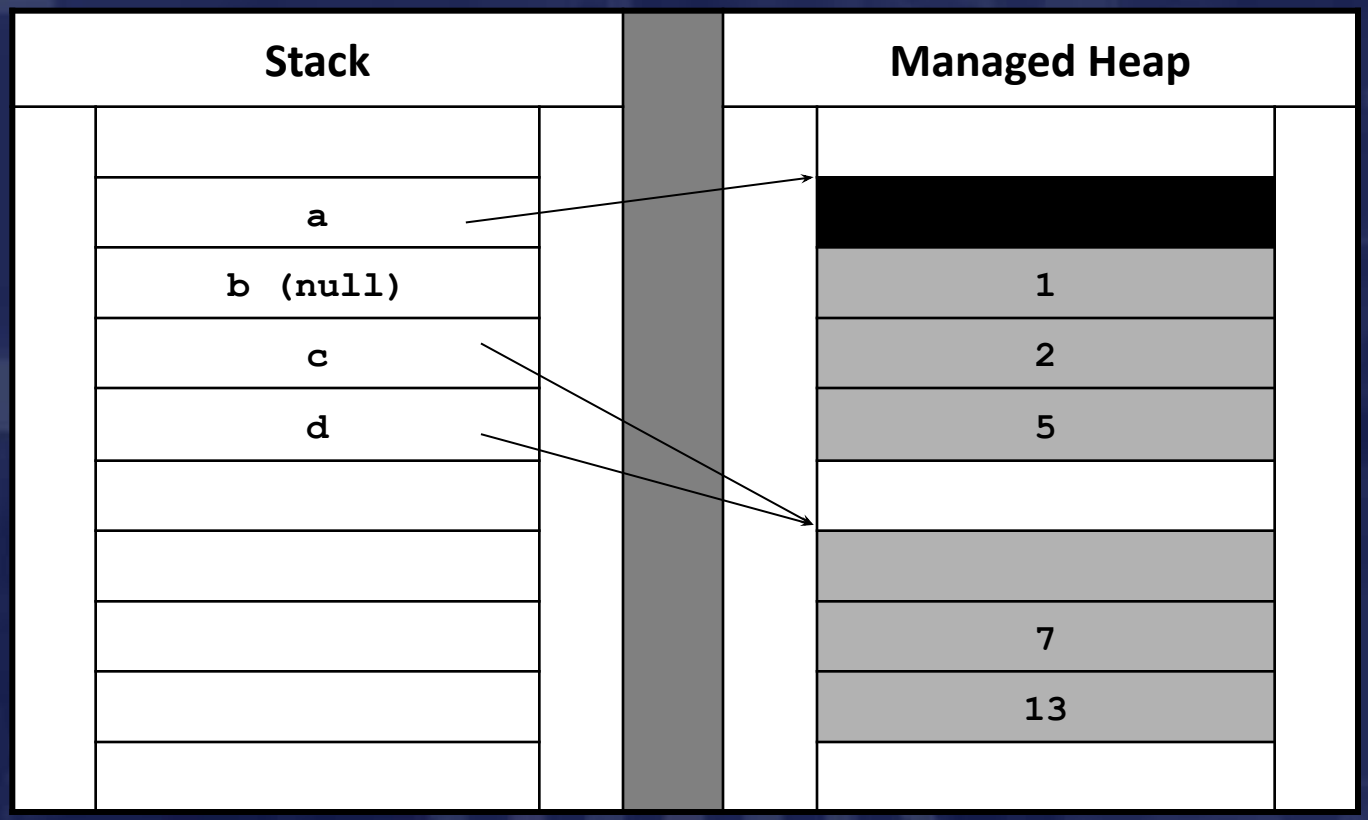
Массивы

- ✔ 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).

- ✔ 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).

Одномерные массивы типов-значений

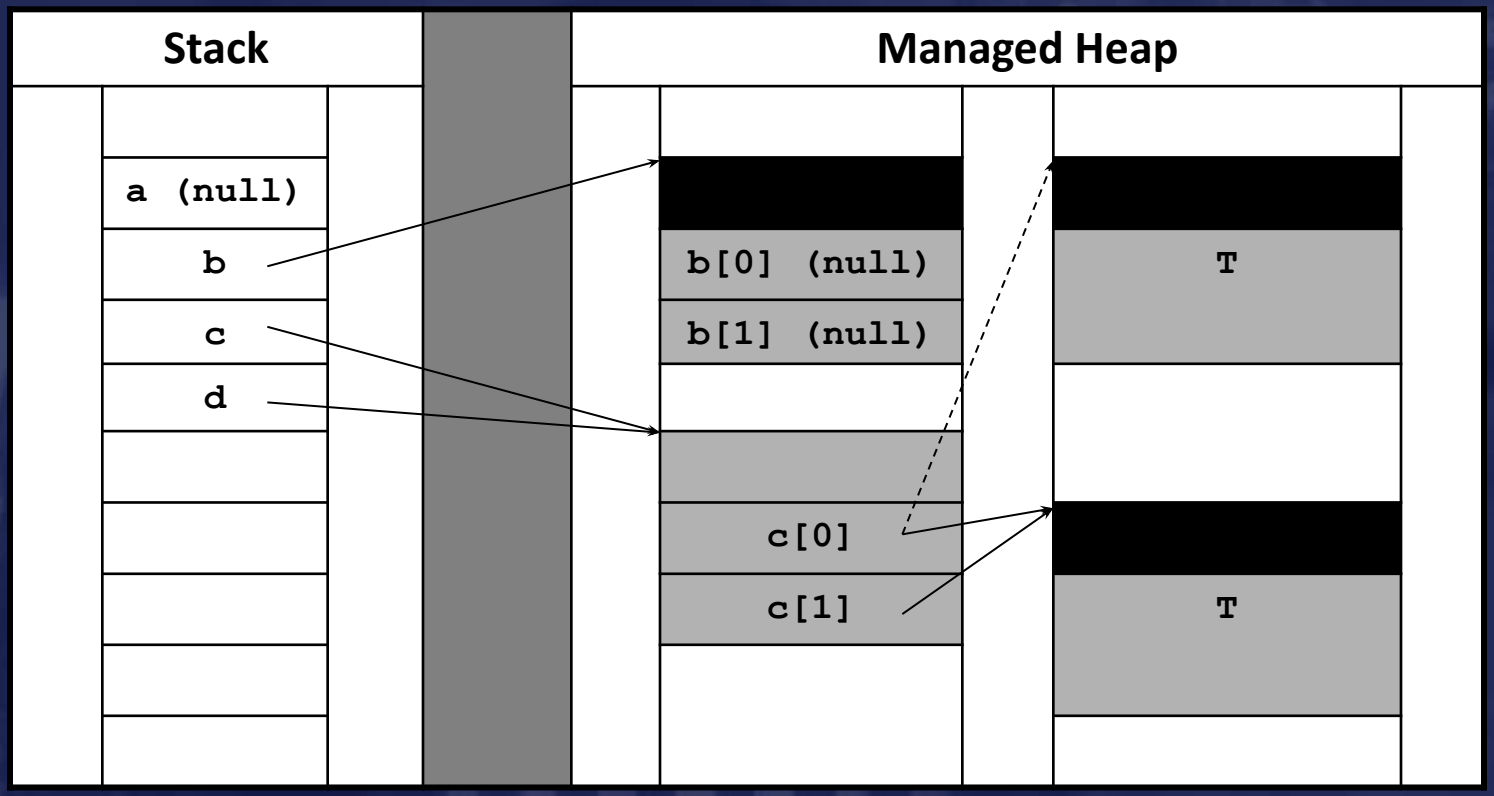
```
int[] a = new int[3] {1,2,5};
int[] b;              // b == null
int[] c = { 7,13 };
int[] d = c;          // d и c – это один и тот же массив!
```



Одномерные массивы ссылочных типов

```
class T
{ ...
  public 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;
```



Инициализация элементов массива Массивы нулевой длины. Приведение типов

- ✔ По умолчанию при создании массива элементы размерных типов инициализируются нулевыми значениями, элементы ссылочных типов –значением null.
- ✔ 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.
```

- ✔ 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 System.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 массива. Метод класса System.Array

```
public object Clone();
```

создает ограниченную(shallow) копию массива - если элементы массива имеют ссылочный тип, то копируются только сами элементы массива (ссылки), но не копируются те объекты, на которые эти ссылки указывают.

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

Копирование массива. Элементы ссылочного типа

```
Book[] books = new Book[2]
 { new Book("C++", "Bjarn Straustroup", 2008),
   new Book("C#", "Эндрю Троелсен", 2005)};

   Console.WriteLine("\nbooks");
   for (int j = 0; j < books.Length; j++)
      Console.WriteLine(books[j]);

   Book[] books_clone = (Book[])books.Clone();
   books_clone[0].Title = "???";

 Console.WriteLine("\nbooks");
 for (int j = 0; j < books.Length; j++)
    Console.WriteLine(books[j]);

 Console.WriteLine("\nbooks_clone");
  for (int j = 0; j < books.Length; j++)
     Console.WriteLine(books_clone[j]);
```

```
class Book
   {
      public string Title { get; set; }
      public string Author { get; set; }
      public int Year { get; set; }

      public Book(string Title, string Author, int Year)
      {
         this.Title = Title;
         this.Author = Author;
         this.Year = Year;
       }
      public override string ToString()
         {    return Author + " " + Title + " " + Year;
         }
   }
```

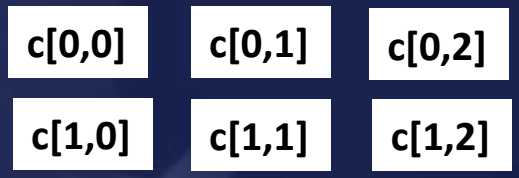
```
books
C++  Bjarn Straustroup  2008
C#  Эндрю Троелсен  2005

books
??? Bjarn Straustroup  2008
C#  Эндрю Троелсен  2005

books_clone
??? Bjarn Straustroup  2008
C#  Эндрю Троелсен  2005
```

Многомерные 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
```

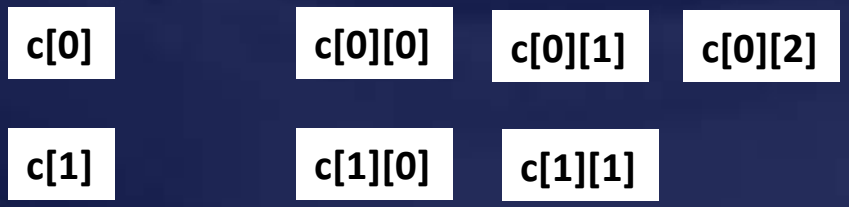


- ✔ Прямоугольные массивы можно копировать при помощи Clone().

Многомерные ступенчатые (jagged) массивы

- ✔ Ступенчатый массив – массив, элементами которого являются массивы, которые могут иметь разную длину.

```
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
```



- ✔ Метод Clone() копирует только c[0] и c[1] (ссылки на одномерные массивы).

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.
- ✔ Посимвольный доступ разрешен только для чтения.
- ✔ 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);

```
string s = "Hello, World!";

Console.WriteLine(s);
Console.WriteLine(s[0]);

for (int j=0; j<s.Length; j++) Console.Write (s[j]);

// s[0] = 'h';        // ошибка
```

- ✔ Вывод:

```
Hello, World!
H
Hello, World!
```

Операции сложения в классе System.String

- ✔ Операция сложения двух строк определена как конкатенация строк
- ✔ Определены операции сложения строк и арифметических типов

```
string s1 = "Hello";
string s2 = "World";
string s3 = s1 + ", " + s2 + "!";
Console.WriteLine(s3);                          // Hello, World!

double d =1234567890.0123456789;
Console.WriteLine(s2 + d);                      // World1234567890.01235
```

Операции сложения в классе System.String

✔ Определена операция сложения строки и объекта типа object. Результат – объект типа string, который получается в результате сложения двух строк - левого операнда и значения, который возвращает виртуальный метод ToString() для правого операнда.

```
class Book
  {   private string Title;
      private string Author;
      private int Year;

    public Book (string Title = "C#",
                 string Author = "Э. Троелсен",
                 int Year = 2013)
      {  this.Title = Title;
         this.Author = Author;
         this.Year = Year;
      }

    public override string ToString()
      {   return Author + " " + Title + " " + Year;
      }
  }
```

```
string s0 = "-----";
Book book = new Book();
Console.WriteLine(s0 + book);
```

Вывод:

```
-----Э. Троелсен C# 2013
```

Составное форматирование и интерполяция строк

✔ В примере для формирования строки с данными объекта типа Book использованы

- операция сложения строк – в методе ToString();
- составное форматирование строк - в методе ToString_FormattedString ();
- интерполяция строк – в методе ToString_InterpolatedString ();

```
class Book
{       private string Title;
        private string Author;
        private int Year;
       public Book (string Title = "C#",  string Author = "Э. Троелсен",  int Year = 2013)
        {   this.Title = Title;
            this.Author = Author;
            this.Year = Year;
         }
        public override string ToString()
         {   return "Автор : " + Author + "\nНазвание : " + Title + "\nГод издания : " + Year;  }

         public string ToString_FormattedString()
         {   return String.Format("Автор : {0}\nНазвание : {1}\nГод издания : {2}", Author, Title, Year);  }

         public string ToString_InterpolatedString()
         {   return $"Автор : {Author}\nНазвание : {Title}\nГод издания : {Year}";      }
   }
```

Метод Split класса System.String

- Метод Split (6 перегрузок) формирует из строки массив строк, используя как разделители заданные символы.
- Память под массив строк распределяется в самом методе Split.

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

- Пример

```
string str = "ab cd;abc; 1234";
string[] sar1 = str.Split(';',' ');
for ( int j = 0; j < sar1.Length; j++)   Console.WriteLine(sar1[j]);

char[] delim = {';'};
string[] sar2 = str.Split(delim,2);
for ( int j = 0; j < sar2.Length; j++)    Console.WriteLine(sar2[j]);
```

Вывод:

```
ab
cd
abc

1234
ab cd
abc; 1234
```

- Определена перегрузка метода Split, которая пропускает пустые строки.

```
public string[] Split( char[] separator, StringSplitOptions options );
```

- Перечисление StringSplitOptions имеет два значения: None и RemoveEmptyEntries.

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;}.

```
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();
```

- ✔ Множество символов строки изменяется с помощью перегруженных методов Remove, Append, AppendFormat, Insert, Replace.
- ✔ В классе определены перегрузки методов для всех арифметических типов и типа object.

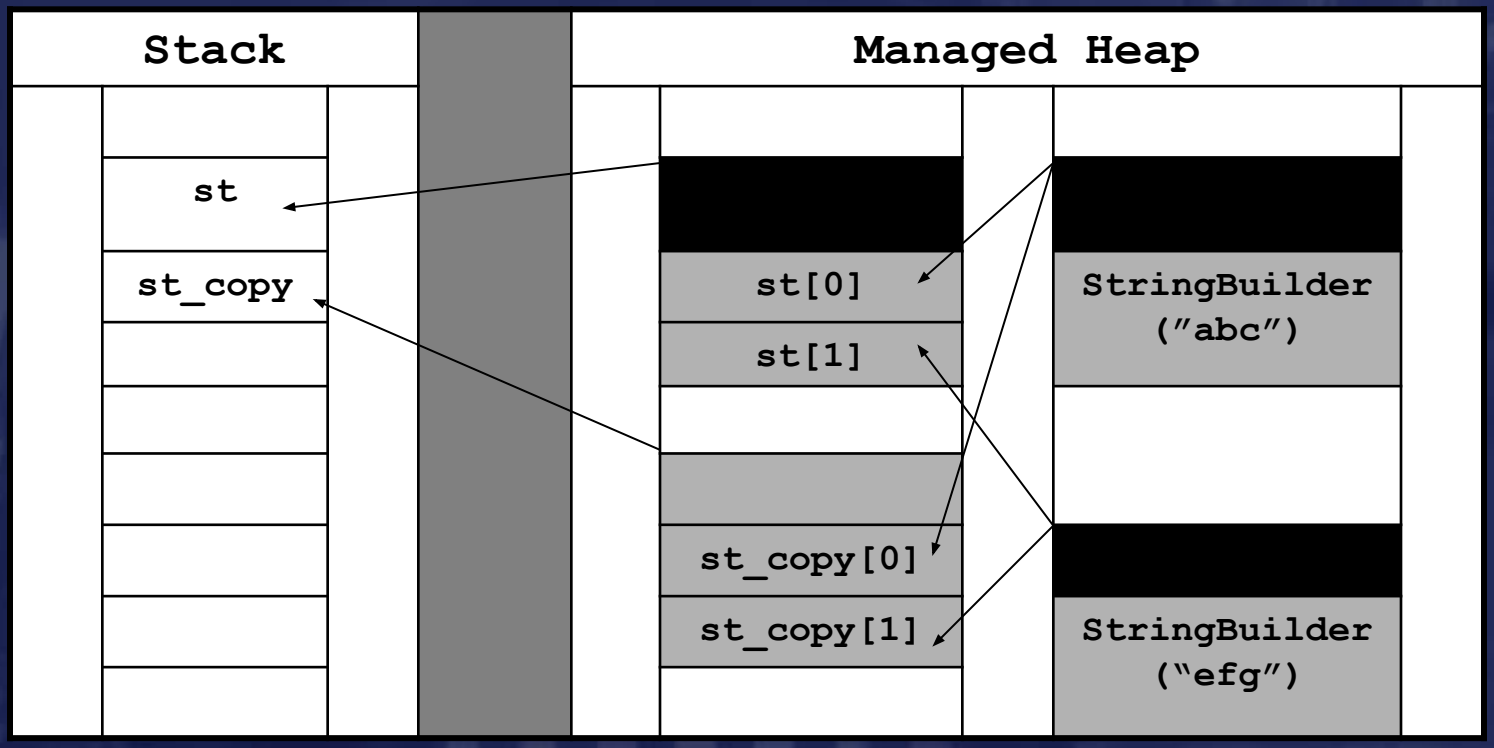
Пример Arrays_Demo

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

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

Console.WriteLine("\nst");
for (int j=0; j<st.Length; j++ ) Console.Write("  {0}", st[j]);                    // abc efz

Console.WriteLine("\nst_copy");
for (int j=0; j<st_copy.Length; j++ ) Console.Write("  {0}", st_copy[j]);          // 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”,”!”);
```

- 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 метки-заполнителя(placeholder) в строке 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 метки-заполнителя(placeholder) в строке форматирования

```
{N,M:F<R>}
```

Количество выводимых разрядов

Формат вывода

Ширина поля

Номер параметра (начинаются с нуля)

```
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
```