Полезные классы .Net

Структура Complex

- **представления комплексного числа**

Комплексное число — это число, включающее в себя вещественное числовую часть и мнимую числовую часть.

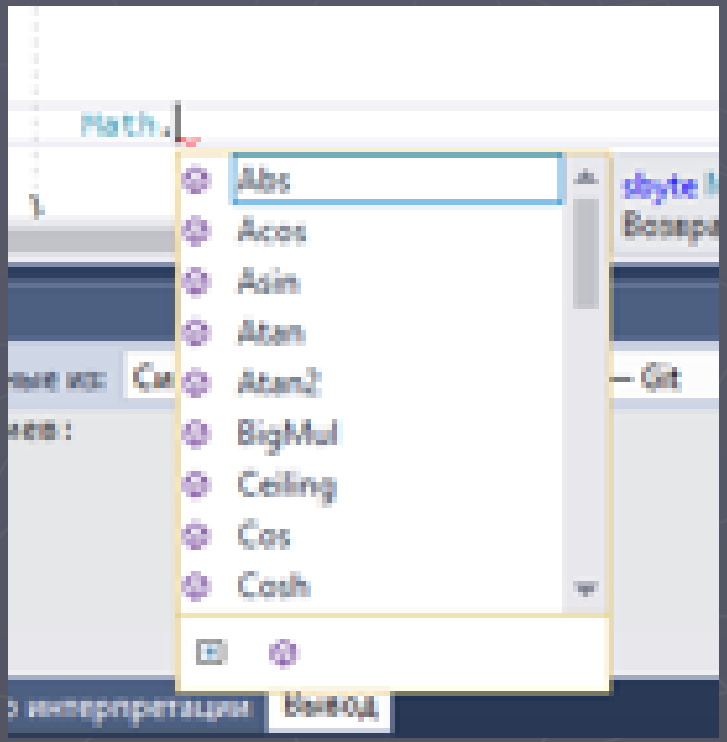
```
Complex z1 = new Complex(3, 5);
Complex z2 = new Complex(-2, 10);
```

System.Math

- набор методов, соответствующих основным математическим функциям

Abs()
Acos(), Asin(), Atan()
Atan2()
BigMul()
Ceiling()
Cos(), Sin(), Tan()
Cosh(), Sinh(), Tanh()
DivRem()
E
Exp()
Floor()

IEEERemainder()
Log()
Log10()
Max(), Min()
PI
Pow()
Round()
Sign()
Sqrt()
Truncate()



System.Random

- генерирует псевдослучайную последовательность значений byte, int или double.

```
//Создание объекта для генерации чисел
Random rnd = new Random();

//Получить очередное (в данном случае - первое) случайное число
int value = rnd.Next();

//Вывод полученного числа в консоль
Console.WriteLine(value);
```

```
1  //Создание объекта для генерации чисел
2  Random rnd = new Random();
3
4  //Получить случайное число (в диапазоне от 0 до 10)
5  int value = rnd.Next(0, 10);
6
7  //Вывод числа в консоль
8  Console.WriteLine(value);
```

```
Random rnd = new Random();

int randomNumber1 = random.Next();

int randomNumber2 = rnd.Next(1, 6);  // генерация случайного чисела от 1 до 5

int randomNumber3 = rnd.Next(91);    // генерация случайного чисела от 0 до 90
```

System.Random

```
public virtual void NextBytes (Span<byte> buffer);
```

Заполняет элементы указанного диапазона байтов случайными числами

```
Random rand = new Random(5);
int intRand = rand.Next() +
              rand.Next(10) +
              rand.Next(-10, 10);
double doubRand = rand.NextDouble();
byte[] buffer = new byte[10];
rand.NextBytes(buffer);
```

вещественное число из интервала [0, 1)

заполняет массив байтов случайными значениям

Структура TimeSpan

- хранит интервал времени в виде отсчётов по 100 наносекунд

```
var t1 = new TimeSpan(50);
// количество отсчётов по 100 нс
var t2 = new TimeSpan(1, 20, 50);
 // часы, минуты, секунды
```

позволяют обратиться либо к отдельному временному компоненту интервала (свойства Days, Hours, Minutes, Seconds, Milliseconds), либо выразить весь интервал через указанную единицу времени (TotalDays, TotalHours и т.п.).

```
TimeSpan ts = TimeSpan.FromDays(5) - TimeSpan.FromSeconds(1);
            Console.WriteLine(ts.Days);         // 4
            Console.WriteLine(ts.Seconds);      // 59
            Console.WriteLine(ts.TotalDays);
                                        // 9.99998842592593
```

DateTime

- предназначена для хранения даты и времени (точность времени – 100 наносекунд)

год, месяц, день, часы, минуты, секунды, миллисекунды и тип времени (Unspecified, Local, Utc).

```
var dt1 = new DateTime(2021, 12, 11);
var dt2 = new DateTime(2021, 12, 11, 12, 0, 0,
                       DateTimeKind.Local);


var dt3 = new DateTime(2021, 12, 11,
                       new PersianCalendar());
```

Определяет календарь –влияет на алгоритм вычисления некоторых свойств даты

- статические свойства Now, Today, UtcNow
- методы для увеличения
- Сравнения и аддитивные операции

```
var dt = DateTime.Today;
   // текущая дата (без времени)
Console.WriteLine(dt.Month);
    // текущий месяц
Console.WriteLine(dt.DayOfWeek);
     // и день недели
var modDt = dt.AddDays(-3.8);

Console.WriteLine(modDt);
```

```
11
Tuesday
23.11.2018 4:48:00
Для продолжения нажми
```

DateTimeOffset

- хранит смещение всемирного координированного времени (UTC offset), - возможность правильно сравнивать даты разных часовых поясов

```
var dtf = new DateTime(2019, 12, 11);
var dto = new DateTimeOffset(
          dtf, TimeSpan.FromHours(-6));
```

Преобразование информации

- Статический класс System.Convert - взаимные преобразования данных базовых типов (числовые типы, bool, string и DateTime)
- Возможные исключения - InvalidCastException, FormatException, OverflowException

```
byte x = Convert.ToByte("100");
                // x = 100
bool y = Convert.ToBoolean(25);
                // y = true
int z = Convert.ToInt32(DateTime.Now);
                // InvalidCastException
```

методы вида ТоИмяТипа(), где ИмяТипа является именем CLR для базовых типов

- Все базовые типы явным образом реализуют интерфейс System.Iconvertible
- набор методов вида ToИмяТипа(), где ИмяТипа – имя CLR для базовых типов

Статический класс System.BitConverter

- содержит набор методов для преобразования переменной числового или булевого типа в массив байт, а также методы обратного преобразования

```
byte[] data = BitConverter.GetBytes(1099);

int x = BitConverter.ToInt32(data, 0);
```

Класс CultureInfo

- поставщик формата

```
// создаём поставщик формата для русской культуры
// список кодов: msdn.microsoft.com/en-us/goglobal/bb896001
var culture = new CultureInfo("ru-Ru");
```

идентификатор культуры

```
var s1 = dt.ToString("D", culture);
```

- Некоторые типы дополнительно перегружают
- Parse() - возвращает преобразованное число

В случае неудачи генерирует исключение

- и TryParse() - возвращает логическое значение, которое указывает, успешно ли выполнено преобразование, и возвращает преобразованное число в параметр out

В случае неудачи возвращает значение false

в качестве параметра принимает строку и возвращает объект текущего типа

```
int a = int.Parse("10");
double b = double.Parse("23,56");
decimal c = decimal.Parse("12,45");
byte d = byte.Parse("4");
Console.WriteLine($"a={a}  b={b}  c={c}  d={d}");
```

```
1  Console.WriteLine("Введите строку:");
2  string? input = Console.ReadLine();
3
4  bool result = int.TryParse(input, out var number);
5  if (result == true)
6      Console.WriteLine($"Преобразование прошло успешно. Число: {number}");
7  else
8      Console.WriteLine("Преобразование завершилось неудачно");
```

Convert

еще один способ для преобразования значений

```
1  int n = Convert.ToInt32("23");
2  bool b = true;
3  double d = Convert.ToDouble(b);
4  Console.WriteLine($"n={n}  d={d}");
```

как и в случае с методом Parse, если методу не удастся преобразовать значение к нужному типу, то он выбрасывает исключение FormatException.

Базовый класс System.Text.Encoding

- поддерживает различные текстовые кодировки и наборы символов

```
Encoding utf8 = Encoding.GetEncoding("utf-8");
 Encoding ascii = Encoding.ASCII;

byte[] asciiBytes = ascii.GetBytes("TEXT");
  string s = ascii.GetString(asciiBytes);
```

для перевода строки или массива символов в массив байтов (коды символов) и обратно

Класс Array

пределяет ряд свойств и методов, которые мы можем использовать при работе с массивами

```
var people = new string[] { "Tom", "Sam", "Bob", "Kate", "Tom", "Alice" };

// находим индекс элемента "Bob"
int bobIndex = Array.BinarySearch(people, "Bob");
// находим индекс первого элемента "Tom"
int tomFirstIndex = Array.IndexOf(people, "Tom");
// находим индекс последнего элемента "Tom"
int tomLastIndex = Array.LastIndexOf(people, "Tom");
// находим индекс первого элемента, у которого длина строки больше 3
int lengthFirstIndex = Array.FindIndex(people, person => person.Length > 3);
// находим индекс последнего элемента, у которого длина строки больше 3
int lengthLastIndex = Array.FindLastIndex(people, person => person.Length > 3);

Console.WriteLine($"bobIndex: {bobIndex}");                 // 2
Console.WriteLine($"tomFirstIndex: {tomFirstIndex}");       // 0
Console.WriteLine($"tomLastIndex: {tomLastIndex}");         // 4
Console.WriteLine($"lengthFirstIndex: {lengthFirstIndex}"); // 3
Console.WriteLine($"lengthLastIndex: {lengthLastIndex}");   // 5
```

Поиск элемента по условию

```
var people = new string[] { "Tom", "Sam", "Bob", "Kate", "Tom", "Alice" };

// находим первый и последний элементы
// где длина строки больше 3 символов
string? first = Array.Find(people, person => person.Length > 3);
Console.WriteLine(first); // Kate
string? last = Array.FindLast(people, person => person.Length > 3);
Console.WriteLine(last); // Alice

// находим элементы, у которых длина строки равна 3
string[] group = Array.FindAll(people, person => person.Length == 3);
foreach (var person in group) Console.WriteLine(person);
// Tom Sam Bob Tom
```

Изменение порядка элементов массива

```
var people = new string[] { "Tom", "Sam", "Bob", "Kate", "Tom", "Alice" }

Array.Reverse(people);

foreach (var person in people)
    Console.Write($"{person} ");
// "Alice", "Tom", "Kate", "Bob", "Sam", "Tom"
```

Изменение размера массива

```
var people = new string[] { "Tom", "Sam", "Bob", "Kate", "Tom", "Alice" };

// уменьшим массив до 4 элементов
Array.Resize(ref people, 4);

foreach (var person in people)
    Console.Write($"{person} ");
// "Tom", "Sam", "Bob", "Kate"
```

Копирование массива

```
var people = new string[] { "Tom", "Sam", "Bob", "Kate", "Tom", "Alice" };

var employees = new string[3];

// копируем 3 элемента из массива people с индекса 1
// и вставляем их в массив employees начиная с индекса 0
Array.Copy(people,1, employees,0, 3);

foreach (var person in employees)
    Console.Write($"{person} ");
// Sam Bob Kate
```

Сортировка массива

```
Array.Sort(people);

foreach (var person in people)
    Console.Write($"{person} ");

// Alice Bob Kate Sam Tom Tom
```

Span

- представляет непрерывную область памяти.

Цель данного типа - повысить производительность и эффективность использования памяти.

- позволяет избежать дополнительных выделений памяти при операции с наборами данных. Поскольку Span является структурой

Создание Span

- `Span()`: создает пустой объект Span
- `Span(T[] array)`: создает объект Span из массива array
- `Span(void* pointer, int length)`: создает объект Span, который получает length байт памяти, начиная с указателя pointer
- `Span(T[] array, int start, int length)`: создает объект Span, который получает из массива array length элементов, начиная с индекса start

```
string[] people = new string[] { "Tom", "Alice", "Bob" };
Span<string> peopleSpan = new Span<string>(people);
```

```
string[] people = new string[] { "Tom", "Alice", "Bob" };
Span<string> peopleSpan = people;
```

```
string[] people = new string[] { "Tom", "Alice", "Bob" };
Span<string> peopleSpan = people;
peopleSpan[1] = "Ann";                  // переустановка значения элемента
Console.WriteLine(peopleSpan[2]);       // получение элемента
Console.WriteLine(peopleSpan.Length);   // получение длины Span

// перебор Span
foreach (var s in peopleSpan)
{
    Console.WriteLine(s);
}
```

Span позволяет работать с памятью более эффективно и избежать ненужных выделений памяти

```
int[] temperatures = new int[]
{
    10, 12, 13, 14, 15, 11, 13, 15, 16, 17,
    18, 16, 15, 16, 17, 14,  9,  8, 10, 11,
    12, 14, 15, 15, 16, 15, 13, 12, 12, 11
};
int[] firstDecade = new int[10];    // выделяем память для первой декады
int[] lastDecade = new int[10];     // выделяем память для второй декады
Array.Copy(temperatures, 0, firstDecade, 0, 10);    // копируем данные в первый массив
Array.Copy(temperatures, 20, lastDecade, 0, 10);    // копируем данные во второй массив
```

```
int[] temperatures = new int[]
{
    10, 12, 13, 14, 15, 11, 13, 15, 16, 17,
    18, 16, 15, 16, 17, 14,  9,  8, 10, 11,
    12, 14, 15, 15, 16, 15, 13, 12, 12, 11
};
Span<int> temperaturesSpan = temperatures;

Span<int> firstDecade = temperaturesSpan.Slice(0, 10);    // нет выделения памяти под данные
Span<int> lastDecade = temperaturesSpan.Slice(20, 10);    // нет выделения памяти под данные
```