Тема 9. Строки.
Методы класса String.

Учебные вопросы:

1. Введение в строки.
2. Основные операции со строками.
3. Отладка программ.

1. Введение в строки.

Строка в программировании — это последовательность символов, представляющих текстовые данные. Строки могут содержать буквы, цифры, пробелы, знаки препинания и другие символы. В большинстве языков программирования, включая C#, строки представлены как массив символов (char), хотя они обычно используются как единое целое.

В C# строки определяются с помощью двойных кавычек:

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

Роль строки в программировании:

- Хранение текстовой информации: Строки используются для хранения и обработки текстовых данных, таких как имена, адреса, сообщения и другие текстовые элементы.
- Взаимодействие с пользователем: Строки часто используются в пользовательских интерфейсах для отображения информации и получения текстовых данных от пользователя. Например, ввод данных через консоль или формы на веб-сайтах.
- Форматирование и вывод данных: Строки позволяют форматировать данные для вывода на экран или в файл. Они могут включать в себя результаты вычислений, данные из базы данных и другие переменные.
- Работа с текстовыми файлами: Текстовые файлы обычно состоят из строк, которые можно читать, записывать и обрабатывать в программах. Это важно для работы с конфигурационными файлами, логами и другими текстовыми данными.
- Манипуляции и анализ данных: В программировании строки часто используются для манипуляции и анализа данных. Это включает в себя операции поиска, замены, разбиения на подстроки, сортировки и фильтрации текстовых данных.
- Передача данных между системами: Строки часто используются для передачи данных между различными системами или программами, особенно в веб-разработке и работе с API, где данные передаются в формате JSON или XML.

Строка - неизменяемый тип данных.

Неизменяемость (immutable) строки в C# означает, что после создания строки её содержимое не может быть изменено. Любая операция, которая, кажется, изменяет строку, на самом деле создаёт новую строку с внесёнными изменениями, а оригинальная строка остаётся неизменной.

```
string original = "Hello";
string modified = original.ToUpper();


Console.WriteLine(original); // Выведет "Hello"
Console.WriteLine(modified); // Выведет "HELLO"
```

Литералы строк.

Литерал — это константное значение, которое непосредственно записывается в код программы. Для строк в C# существуют два основных типа литералов: **стандартные** и **дословные**.

Стандартные строковые литералы

Ограничители: Окружаются двойными кавычками ("").

Особые символы: Для представления специальных символов, таких как табуляция, новая строка или двойная кавычка внутри самой строки, используются escape-последовательности, начинающиеся с обратного слеша (\)

Литералы строк.

Литерал — это константное значение, которое непосредственно записывается в код программы. Для строк в C# существуют два основных типа литералов: **стандартные** и **дословные**.

Стандартные строковые литералы

Ограничители: Окружаются двойными кавычками ("").

Особые символы: Для представления специальных символов, таких как табуляция, новая строка или двойная кавычка внутри самой строки, используются escape-последовательности, начинающиеся с обратного слеша (\)

Некоторые escape-последовательности

Последова-тельность	Описание
\\	Обратный слеш. Выводит один знак обратного слеша
\'	Апостроф, или одиночная кавычка. Выводит один апостроф
\"	Двойные кавычки. Выводит одну такую кавычку
\n	Пустая строка. Перемещает курсор в начало следующей строки
\t	Горизонтальный отступ – символ табуляции. Перемещает курсор вправо на один отступ

Примеры стандартных строковых литералов:

```
Console.WriteLine("Hello, World!");              // Hello, World!
Console.WriteLine("C:\\Program Files\\MyApp");   // C:\Program Files\MyApp
Console.WriteLine("Line 1\nLine 2\nLine 3");     // Line1
                                                 // Line2
                                                 // Line3
Console.WriteLine("\tЭто табуляция");            //         Это табуляция
```

Дословные строковые литералы

Дословные (verbatim) строковые литералы в C# начинаются с символа @ перед двойными кавычками. Такие литералы позволяют записывать строки "как есть", без необходимости использовать управляющие последовательности для специальных символов, таких как обратная косая черта или новая строка.

Особенности дословных строковых литералов:

- Не требуется экранирование символов: В дословных строках обратные косые черты не требуют двойного указания.
- Поддержка многострочных строк: Дословные литералы могут содержать текст, который распространяется на несколько строк.

Дословные строковые литералы

Дословные (verbatim) строковые литералы в C# начинаются с символа @ перед двойными кавычками. Такие литералы позволяют записывать строки "как есть", без необходимости использовать управляющие последовательности для специальных символов, таких как обратная косая черта или новая строка.

Особенности дословных строковых литералов:

- Не требуется экранирование символов: В дословных строках обратные косые черты не требуют двойного указания.
- Поддержка многострочных строк: Дословные литералы могут содержать текст, который распространяется на несколько строк.
- Двойные кавычки внутри строки должны быть удвоены, чтобы быть частью строки.

Примеры дословных строковых литералов

```
Console.WriteLine(@"C:\Program Files\MyApp"); //C:\Program Files\MyApp
Console.WriteLine(@"line 1
                    line 2
                    line 3");
Console.WriteLine(@"Line1\nLine2\nLine3");    //Line1\nLine2\nLine3
Console.WriteLine(@"Hotel ""California""");   //Hotel "California"
```

2. Основные операции со строками

Класс **String** в C# предоставляет множество методов для работы со строками. Ниже приведены некоторые из наиболее часто используемых методов:

Интерполяция строк: Вставка значений переменных в строку с использованием символа $.

```
int age = 30;
string name = "Alice";
string message = $"Name: {name}, Age: {age}";
```

Свойство **Length**: Возвращает количество символов в строке

```
string text = "Hello";
int length = text.Length; // length = 5
```

Метод **Substring**: Извлекает подстроку, начиная с указанного индекса

```
string text = "Hello, World!";
string sub = text.Substring(7, 5); // sub = "World"
```

Методы **ToUpper** и **ToLower**: Преобразуют все символы строки в верхний или нижний регистр.

```
string text = "Hello";
string upperText = text.ToUpper(); // "HELLO"
string lowerText = text.ToLower(); // "hello"
```

Метод **Trim**: Удаляет пробелы в начале и в конце строк

```
string text = "  Hello, World!  ";
string trimmedText = text.Trim(); // "Hello, World!"
```

Метод **IndexOf**: Возвращает индекс первого вхождения подстроки.

```
string text = "Hello, World!";
int index = text.IndexOf("World"); // index = 7
```

Метод **Contains**: Проверяет, содержит ли строка указанную подстроку

```
bool contains = text.Contains("World"); // true
```

Метод **Replace**: Заменяет все вхождения одной подстроки на другую.

```
string text = "Hello, World!";
string replacedText = text.Replace("World", "C#"); // "Hello, C#!"
```

Метод **Split**: Разделяет строку на массив подстрок, используя указанный разделитель

```
string text = "apple,banana,cherry";
string[] fruits = text.Split(','); // ["apple", "banana", "cherry"]
```

Метод **Equals**: Сравнение строк с учетом и без учета регистра

```
string string1 = "Hello";
string string2 = "hello";
// сравнение с учетом регистра
bool result = string1.Equals(string2);
Console.WriteLine(result);  // false
// сравнение без учета регистра
result = string1.Equals(string2, StringComparison.OrdinalIgnoreCase);
Console.WriteLine(result);  // true
```

Метод **ToCharArray**: Преобразует строку в массив символов

```
string text = "Hello";
char[] characters = text.ToCharArray(); // ['H', 'e', 'l', 'l', 'o']
```

Метод **String.Join()** используется для объединения элементов массива или коллекции в одну строку, при этом элементы разделяются указанным разделителем.

```
string[] words = { "apple", "banana", "cherry" };

string result = String.Join(", ", words);

Console.WriteLine(result); // "apple, banana, cherry"
```

Метод **ToString**: Преобразует числовые и другие типы данных в строку

```
int number = 123;
string numberString = number.ToString(); // "123"
bool isTrue = false;
string isTrueString = isTrue.ToString(); // False
```

3. Отладка программ

Отладка - это процесс обнаружения и исправления ошибок в программном коде.

Принято выделять три вида ошибок:

- синтаксические,
- логические,
- исключения.

- **Синтаксические ошибки** программисту искать не нужно. Такие ошибки компилятор находит и подсвечивает красным цветом.
- **Логические ошибки** компилятор не умеет отслеживать. Программисту придётся отыскивать их самостоятельно. Логические ошибки появляются, когда разработчик неправильно описывает алгоритм работы программы.
- **Исключения** (от англ. Exceptions) — это вид ошибок, какой появляется в тех случаях, когда в ходе работы программы возникает ситуация, которую разработчик не описал в коде. То есть программа запускается, но не выполняется целиком.

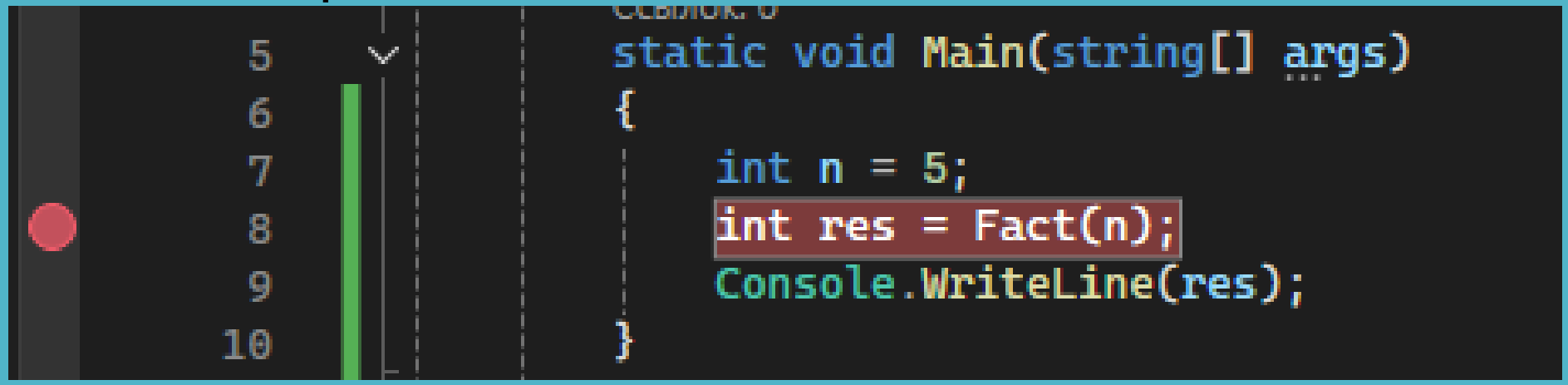
Основные возможности отладки в Visual Studio

1. Точки останова (Breakpoints)

Точки останова позволяют приостановить выполнение программы в определённом месте кода, чтобы вы могли изучить текущее состояние программы.

Установка точки останова: Щелкните на левом поле рядом с номером строки или нажмите F9, находясь на нужной строке кода. Появится красная точка, обозначающая точку останова.

Удаление точки останова: Щелкните на красной точке снова или нажмите F9 ещё раз.



2. Пошаговое выполнение (Step Into, Step Over, Step Out)

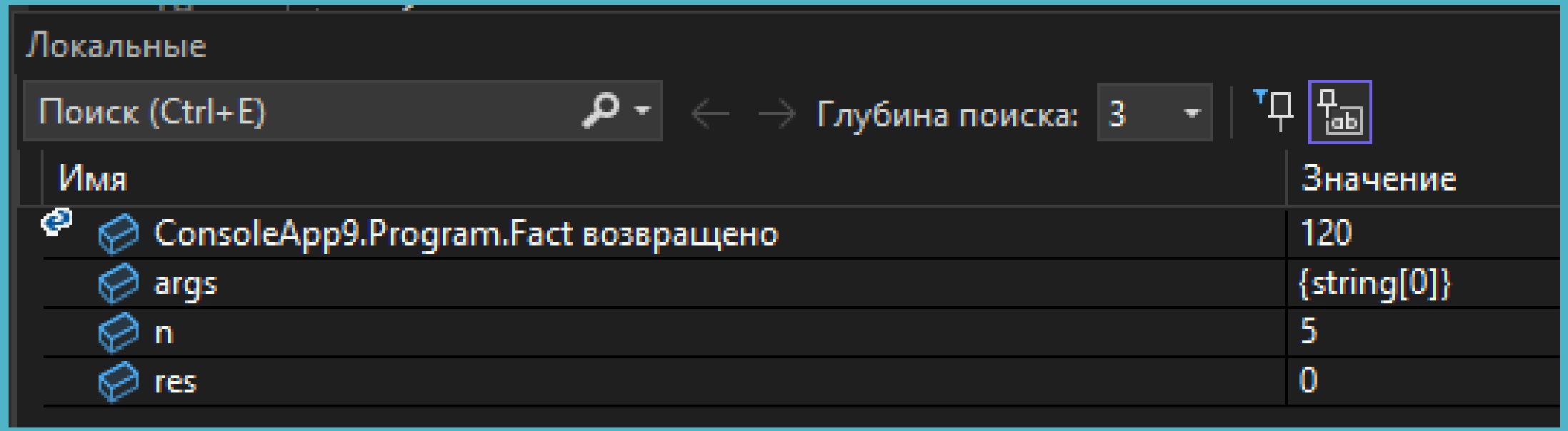
Пошаговое выполнение позволяет вам выполнить код строка за строкой, чтобы увидеть, как он работает.

- Step Into (Шаг внутрь, F11): Переходит внутрь вызываемого метода или выражения.
- Step Over (Шаг поверх, F10): Выполняет текущую строку кода и переходит к следующей, не заходя внутрь методов.
- Step Out (Шаг наружу, Shift + F11): Завершает выполнение текущего метода и возвращается к строке, из которой он был вызван.

3. Просмотр значений переменных

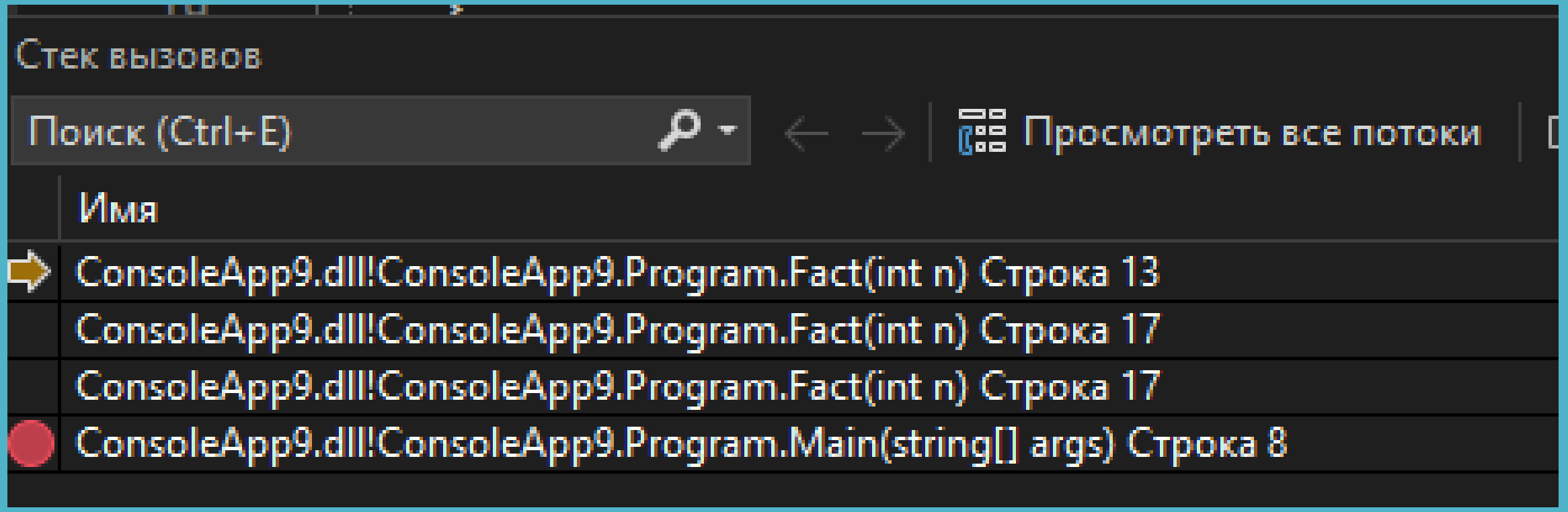
Во время отладки вы можете наводить указатель мыши на переменные, чтобы увидеть их текущее значение.

Окно "Локальные": Автоматически показывают значения переменных в текущем контексте.



4. Call Stack (Стек вызовов)

Стек вызовов показывает вам последовательность вызовов функций, которые привели к текущей точке выполнения программы. Это полезно для понимания того, как ваша программа достигла текущего состояния.



```
static void FirstFunction()
{
    SecondFunction();
}

Ссылок: 1
static void SecondFunction()
{
    int zero = 0;
    int result = 10 / zero; // Ошибка: деление на ноль
}
Ссылок: 0
static void Main(string[] args)
{
    FirstFunction();

}
```

```
MyApp.dll!Program.SecondFunction() Строка 12
MyApp.dll!Program.FirstFunction() Строка 6
MyApp.dll!Program.Main(string[] args) Строка 16
```

Советы по эффективной отладке

- Разбивайте задачу на мелкие части: Отлаживайте небольшие фрагменты кода по отдельности.
- Используйте точки останова эффективно: Ставьте точки останова в ключевых местах, чтобы сузить область поиска ошибок.
- Проверяйте значения переменных: Убедитесь, что значения переменных соответствуют вашим ожиданиям.

Заключение.

- Класс String множество методов и свойств для работы со строками
- В C# строки неизменяемы, любые операции над ними создают новую строку.
- Строковая интерполяция ($"{...}"): Упрощает форматирование строк, позволяя встраивать переменные непосредственно в строку.
- Отладка в Visual Studio — это мощный процесс, который помогает разработчикам находить и исправлять ошибки в коде

Контрольные вопросы:

- Что такое строка в программировании и как она представлена в языке C#?
- Почему строки в C# считаются неизменяемыми? Объясните принцип неизменяемости.
- Какие операции можно выполнять со строками в C#? Приведите примеры.
- Что такое строковая интерполяция и как она используется в C#?
- Объясните разницу между стандартными и дословными строковыми литералами.
- Какой метод используется для разделения строки на массив подстрок? Приведите пример его использования.
- Какие методы класса String позволяют изменить регистр символов в строке?
- Что такое точки останова и как они используются в процессе отладки в Visual Studio?
- Какие виды ошибок можно встретить в процессе отладки программ? Как их можно обнаружить?
- Что такое стек вызовов и как он помогает в отладке программ?

Домашнее задание:

1. Повторить материал лекции.
2. Выполнить задания: Учебное пособие, с.104.

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com

Список литературы:

1. Жаксыбаева Н.Н. Основы объектно-ориентированного программирования: язык C#. Часть 1./ Учебное пособие предназначено для учащихся технического и профессионального образования, Алматы, 2010,

2. https://learn.microsoft.com/ru-ru/dotnet/api/system.string?view=net-8.0