1. **.Net Framework** – платформа, которая создана ***Microsoft*** для разработки приложений. Он состоит из следующих компонентов:

- Common Language Runtime (CLR)(виртуальная машина): среда выполнения, отвечает за выполнение кода, сборку мусора, проверка безопасности кода и управление памятью (выполнение IL (промежуточного кода)).

- Framework Class Library (FCL)/BCL(Base Class Library): библиотеки классов .NET.(веб-службы)

- Common Intermediate Language (CIL): промежуточный байт-код, в который компилируется исходный код .NET.

- Common Language Infrastructure (CLI): Стандарт, описывающий архитектуру и правила для создания, компиляции и выполнения .NET-кода.

IL (Intermediate Language) - промежуточный язык, на котором компилируются программы .NET для выполнения в CLR.

3. JIT-компилятор - компонент CLR, который компилирует код на IL в машинный код во время выполнения программы. Это позволяет повысить производительность, так как машинный код выполняется быстрее, чем IL.

4. CTS (Common Type System) - это стандарт, обеспечивающий единообразие типов между разными языками .NET.

5. \*\*System.Object\*\* определяет базовое поведение для всех объектов. Он имеет методы, такие как Equals, GetHashCode, GetType, и ToString, которые наследуются всеми другими типами в .NET.

6. \*\*mscorlib.dll\*\* - это основная сборка, содержащая базовые классы и типы .NET., такие как System.Object, System.String и др.

7. Сборка - абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов.

8. Существуют два основных вида сборок:

- \*\*Сборки приложений (Application Assemblies)\*\*: сборки, созданные для конкретных приложений. Они могут быть исполняемыми (EXE) или библиотеками (DLL).

- \*\*Сборки библиотек (Library Assemblies)\*\*: сборки, созданные для переиспользования кода. Могут быть использованы другими приложениями.

9. \*\*Assembly Manifest\*\* - это часть метаданных(двоичный набор таблиц данных) в сборке(ключевой компонент сборки), которая содержит инфу о сборке (имя сборки), версию, зависимостях, авторе, список всех файлов и др.

10. \*\*GAC (Global Assembly Cache)\*\* - это централизованное хранилище сборок в .NET. Он позволяет разделять и переиспользовать сборки между разными приложениями.

11. \*\*Managed code\*\* - это код, который выполняется в среде с управлением CLR и подвергается автоматическому управлению памятью. \*\*Unmanaged code\*\* - это код, который выполняется вне среды с управлением и обычно не подвергается автоматическому управлению памятью. (написан на С и С++)

12. Метод \*\*Main\*\* - точка входа для приложения .NET. Он вызывается при запуске приложения и используется для начальной инициализации и выполнения программы.

13. Директива \*\*using\*\* в C# используется для объявления и использования пространств имен.

using System; // Импорт пространства имен System

using MyNamespace; // Импорт пользовательского пространства имен

14. Сборки и пространства имен связаны так: сборки могут содержать одно или несколько пространств имен, и типы определены в этих пространствах имен могут быть доступны внутри сборки и из других сборок, если они добавлены в ссылки.

15. Примитивные - типы данных, которые поддерживаются компилятором напрямую.

- Целочисленные типы: int, long, short, byte, sbyte, uint, ulong, ushort.

- Типы с плавающей точкой: float, double.

- Логический тип: bool.

- Символьный тип: char.

- Типы для работы с текстом: string.

16. \*\*Ссылочные типы\*\* - это типы данных, которые хранят ссылку на объект в памяти, а не на сам объект. Они включают классы, интерфейсы, делегаты, массивы, строки.

17. \*\*Типы-значения\*\* - это типы данных, которые хранят сами значения, а не ссылку на объект. Они включают простые типы, структуры и перечисления (enum).

18. Основное отличие между ссылочными и значимыми типами данных заключается в том, как они хранят данные. Ссылочные типы хранят ссылку на объект в куче (heap), тогда как значимые типы хранят само значение в стеке (stack) или внутри другого объекта (например, внутри класса или структуры). Типы значения легче ссылочных, тк для них не нужно выделять память в куче, их не затрагивает сборки мусора, к ним нельзя обратиться через указатель.

19. \*\*Упаковка (boxing)\*\* - это процесс преобразования значимого типа данных в ссылочный тип (object). \*\*Распаковка (unboxing)\*\* - это процесс обратного преобразования объекта (object) обратно в значимый тип данных. Упаковка и распаковка снижают производительность приложения.

Назначение:

* позволяет использовать типы-значения в коллекциях ( где элементы являются элементами типа object)
* внутренний механизм, который обеспечивает возможность вызывать для типов-значений, подобных int и struct, методы Object.

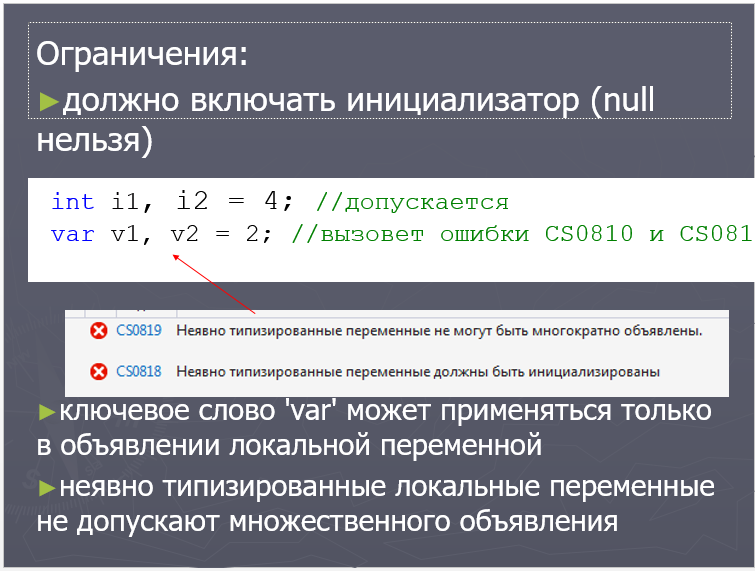
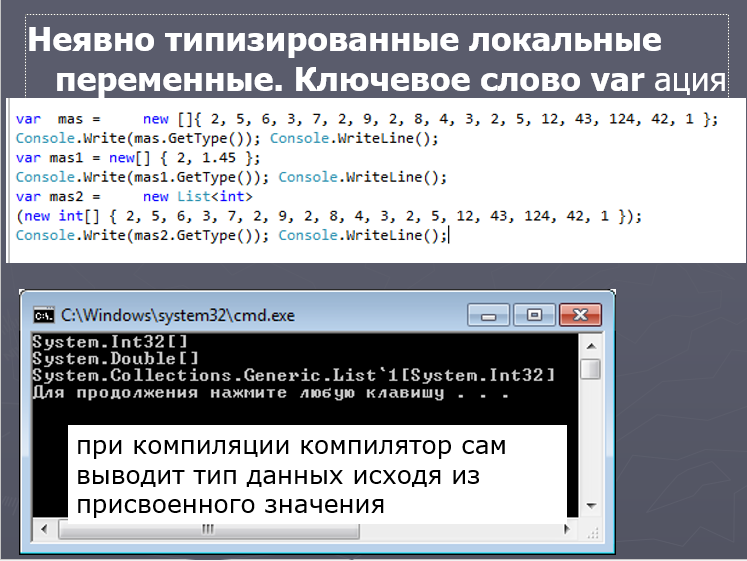


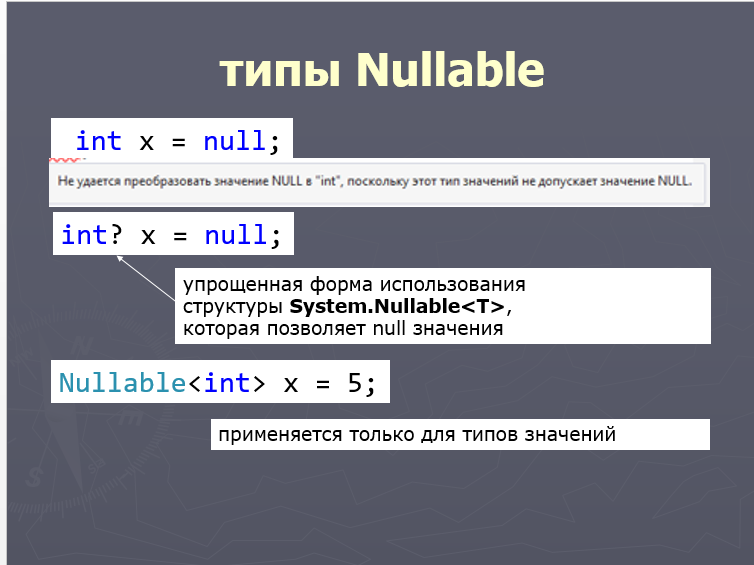
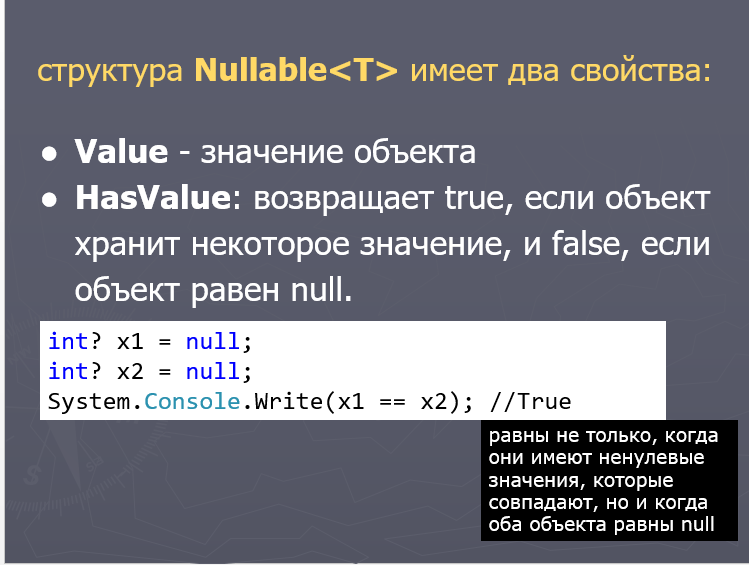
20. Разница между `int` и `System.Int32`, а также `double` и `System.Double`, заключается в том, что `int` и `double` - это синонимы имен типов, предоставляемых C#, в то время как `System.Int32` и `System.Double` - это полные имена типов из пространства имен System.

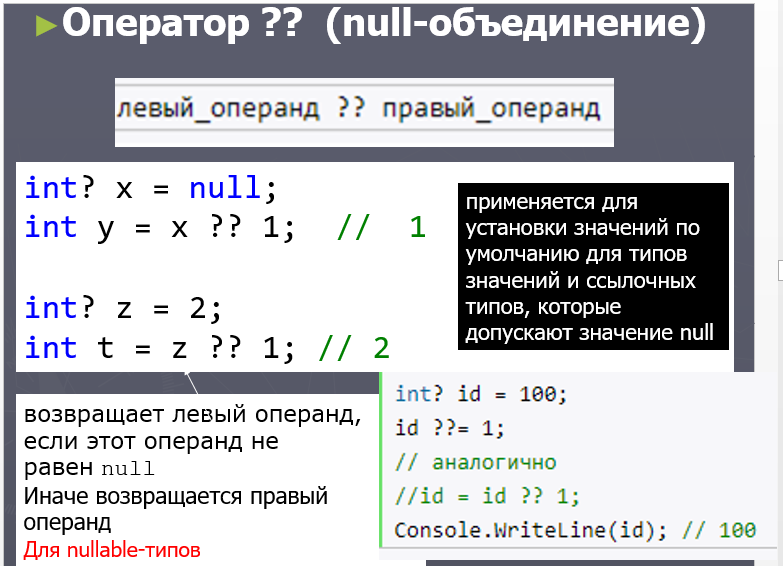
21. Тип `dynamic` представляет собой объект, операции которого будут размещаться во время выполнения. может получить какое угодно начальное значение, и на протяжении времени его существования это значение может быть заменено новым. **НО!!!! корректность указываемых членов компилятором не проверяется**!

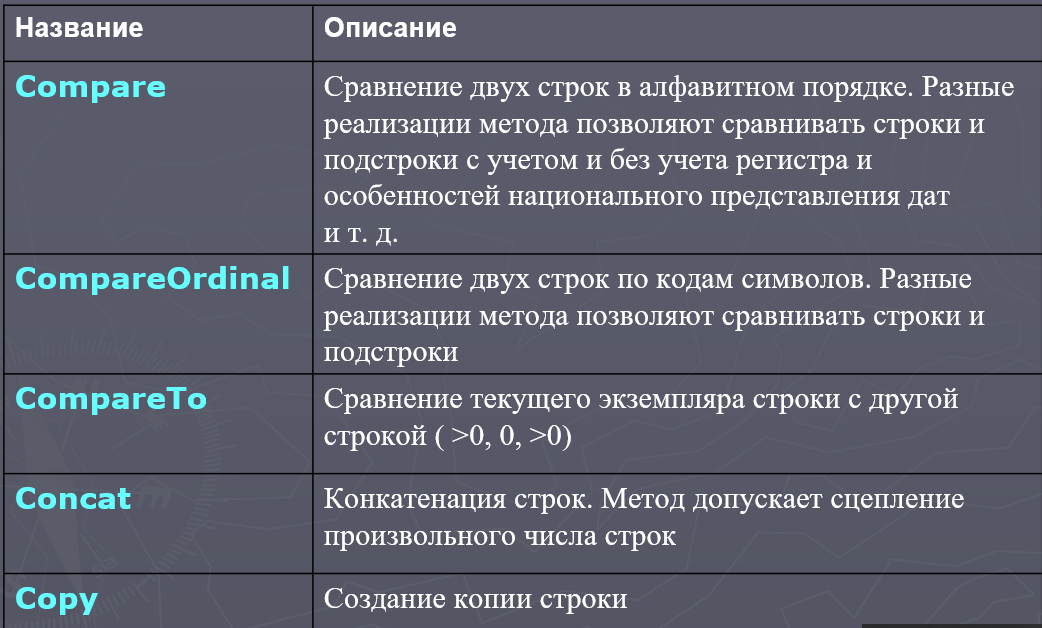
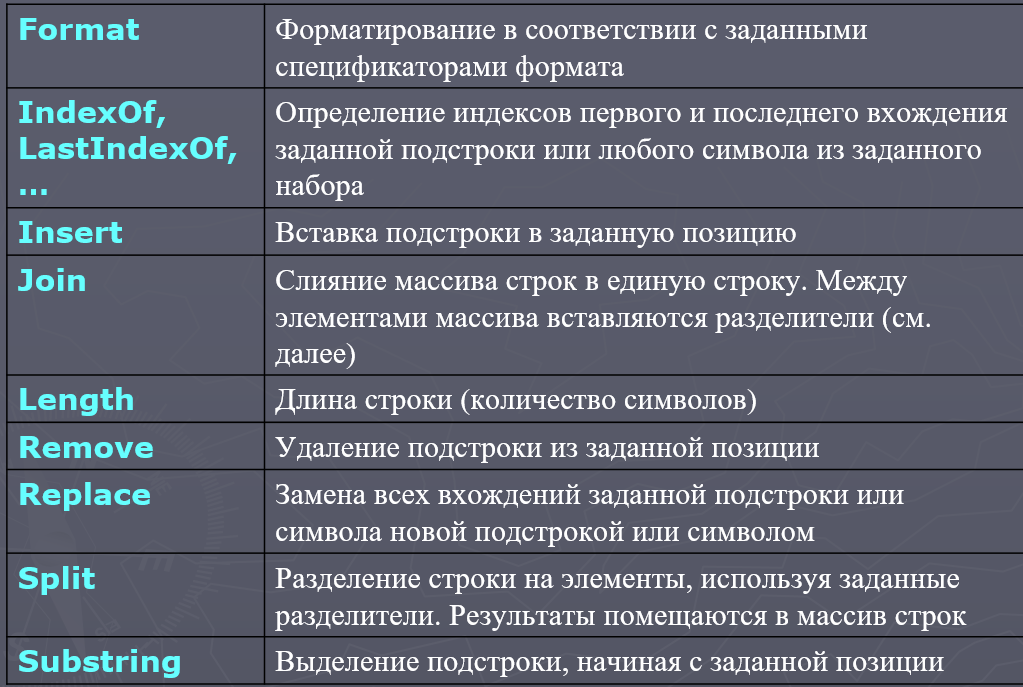
22. Основное отличие между `var` и `dynamic` заключается в том, что `var` используется для объявления переменных с неявной типизацией на этапе компиляции (статическая типизация), в то время как `dynamic` используется для переменных с динамической типизацией на этапе выполнения.

23. \*\*Неявно типизированная переменная\*\* - это переменная, объявленная с использованием ключевого слова `var`, которое позволяет компилятору определить тип переменной на основе ее инициализации. Уменьшает объем кода и улучшает читаемость.



24. \*\*Nullable тип\*\* (типы-значения с `?`) используется для хранения значений, которые могут быть равны `null`. Он часто используется для числовых данных, чтобы указать, что переменная может быть пустой. 



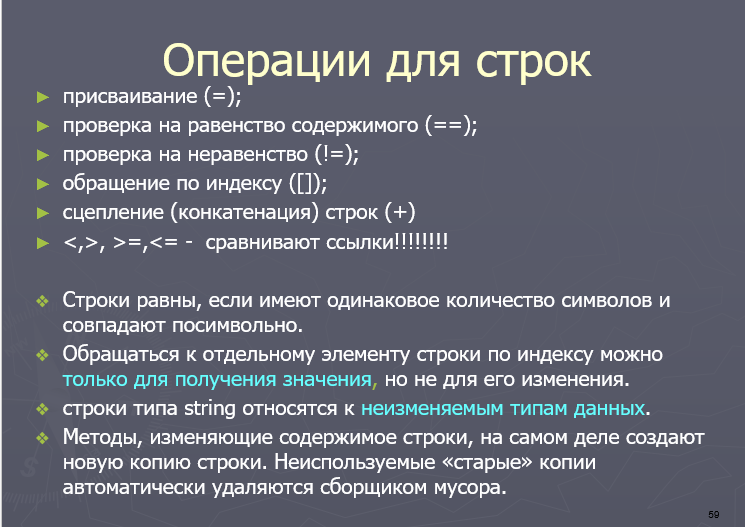
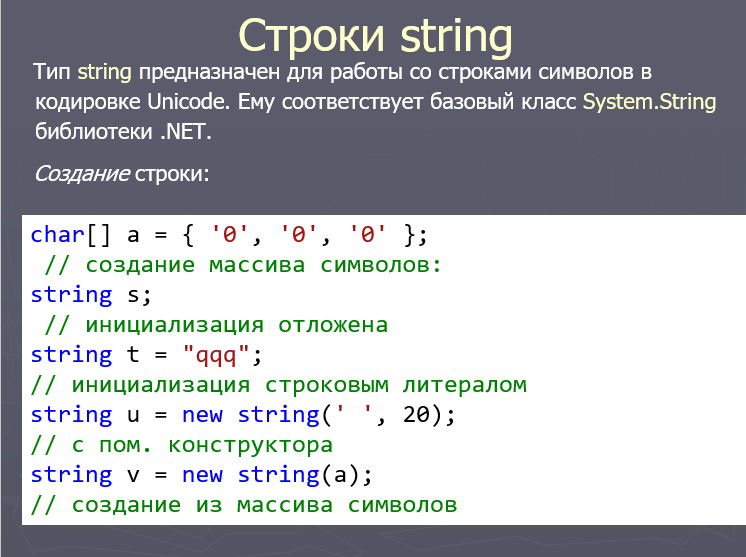
25. Строковый литерал объявляется "Hello, World!".  

26. Строки можно задавать и инициализировать следующими способами:

- С использованием строковых литералов: `string str = "Hello, World!";`

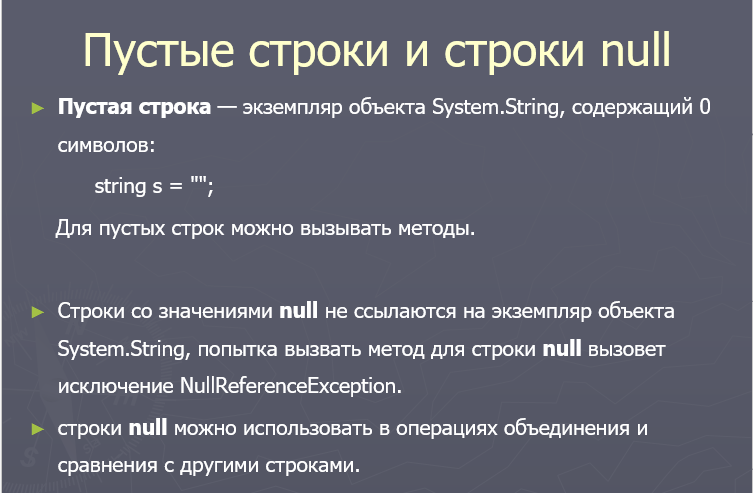
- С использованием конструктора: `string str = new string('A', 5);`

-С использованием интерполяции строк: string name = 'John'; string greeting = $’Hello, {name}!’;



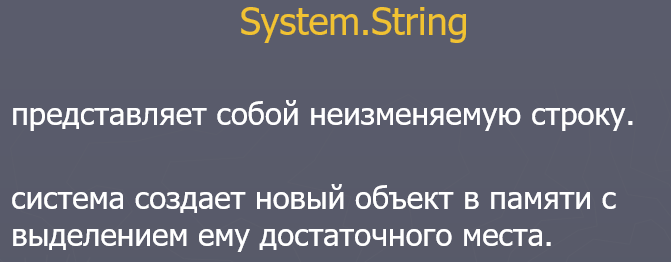
27. выше методы строки

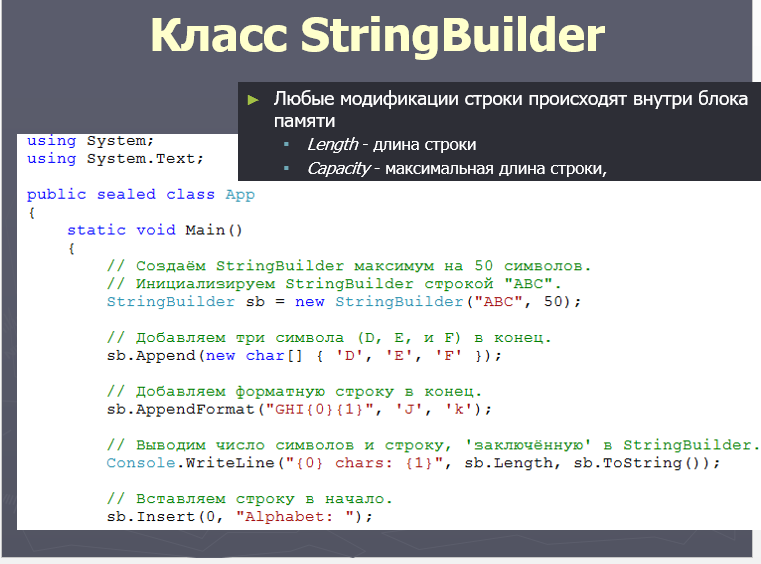
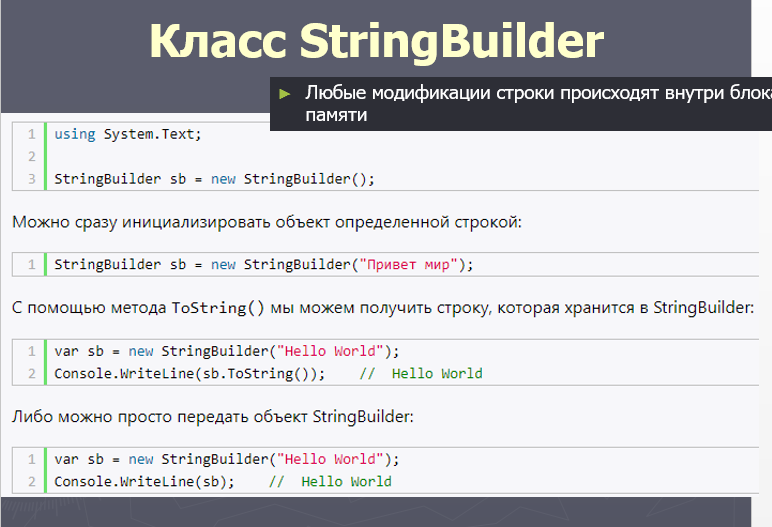
28. \*\*Пустая строка\*\* - экземпляр объекта System.String, содержащий 0 символов. Для пустых строк можно вызывать методы. Null – ссылка на строку, которая не указывает на какой-либо обьект строки. Строки со значениями **null** не ссылаются на экземпляр объекта System.String, попытка вызвать метод для строки **null** вызовет исключение NullReferenceException. строки **null** можно использовать в операциях объединения и сравнения с другими строками.

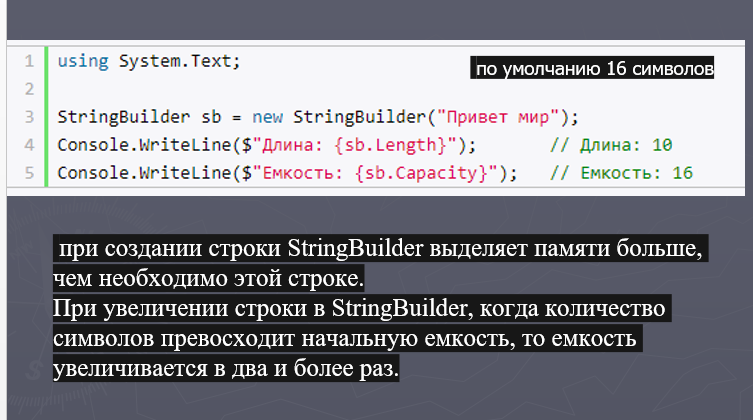
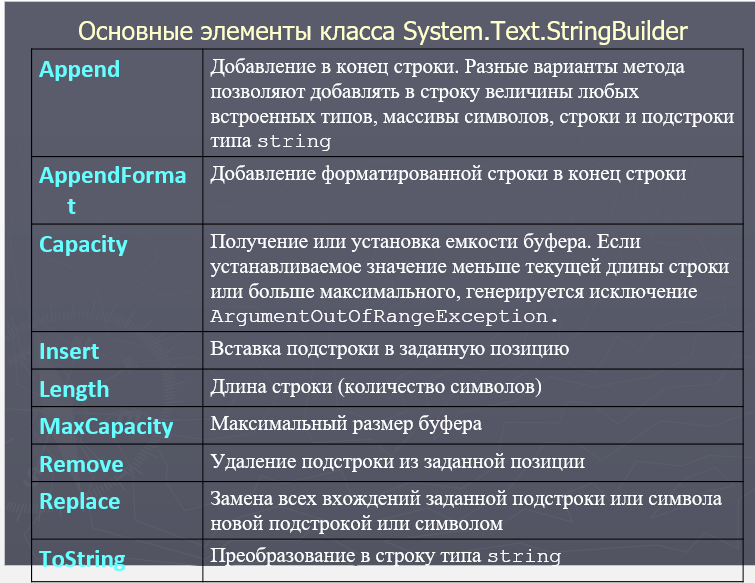


29. Сравнение строк можно выполнить с использованием операторов `==` (сравнение по значению) и `Equals` (сравнение по содержимому), `Compare` и `CompareTo` для определения относительного порядка строк.

30. \*\*String\*\* представляет неизменяемую (immutable) строку, что означает, что каждая операция изменения строки создает новый экземпляр строки, в то время как \*\*StringBuilder\*\* представляет изменяемую строку, позволяя эффективно выполнять множество операций изменения.







31. \*\*Методы Convert\*\* используются для преобразования значений между различными типами данных, например, из строки в число или из числа в строку.

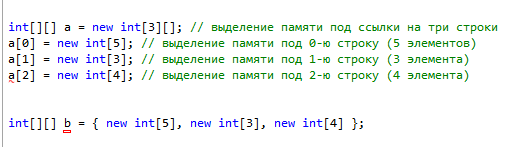
32`Console.WriteLine("Hello, World!");` выводит текст, а `Console.ReadLine()` читает строку из консоли.

33. Примеры определения и инициализации одномерного и двумерного массивов:

- Одномерный массив: `int[] numbers = new int[5];`

- Двумерный массив: `int[,] matrix = new int[3, 3];`

34. \*\*Ступенчатый массив\*\* - это массив массивов, в котором каждый внутренний массив может иметь разную длину. Он создается с помощью вложенных массивов разной длины.

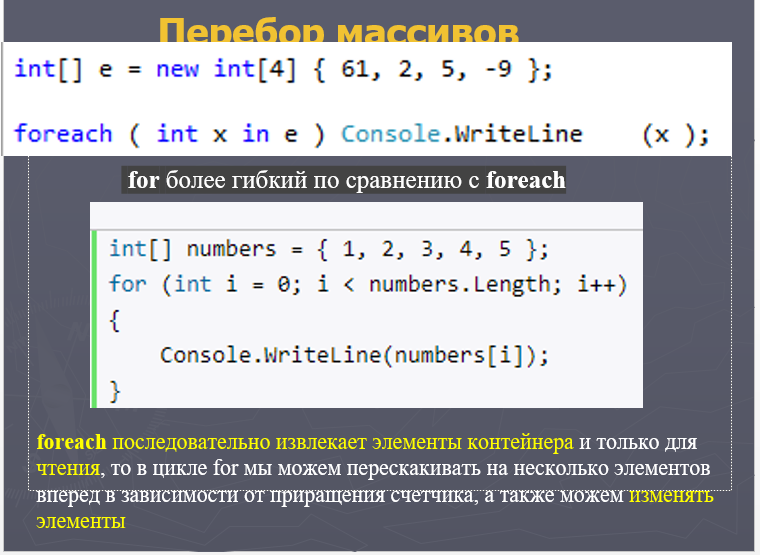


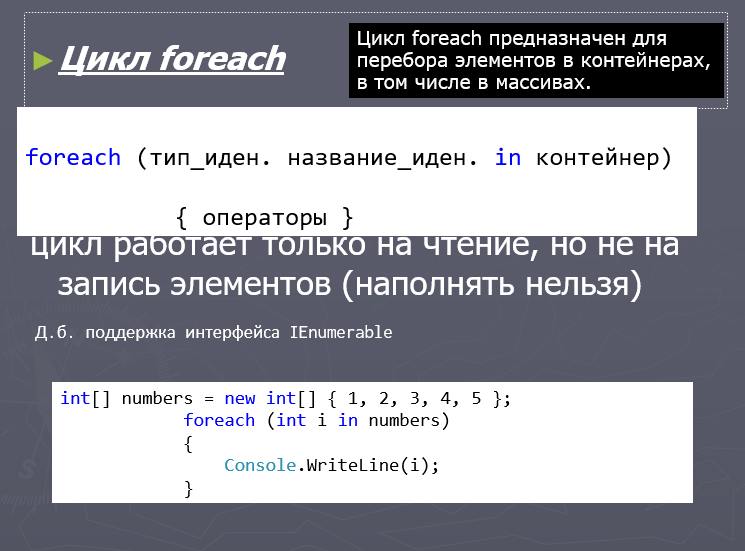
35. В цикле `foreach` можно использовать объекты, реализующие интерфейс `IEnumerable`, такие как массивы, списки и другие коллекции. Пример:

Int[] myArray = {1,2,3,4,5};

foreach (int item in myArray)

{Console.WriteLine(item);} цикл работает только на чтение, но не на запись элементов

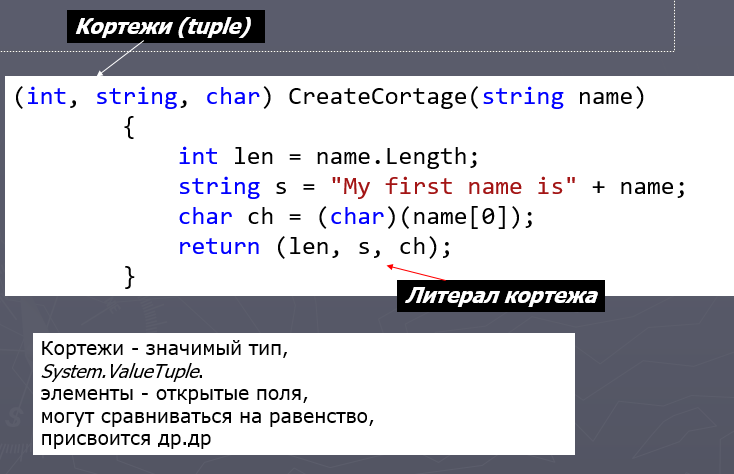
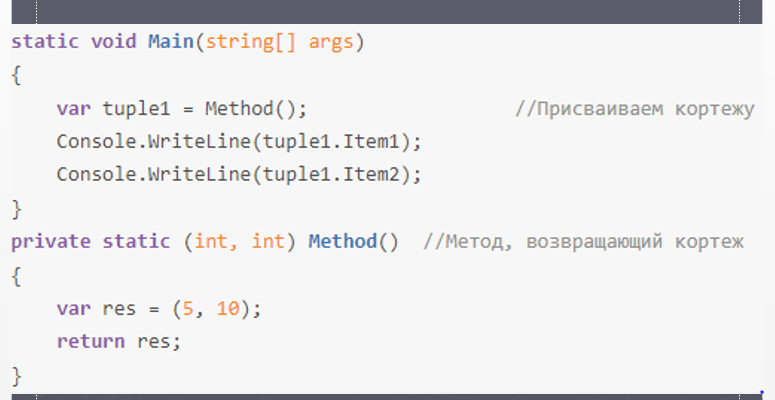
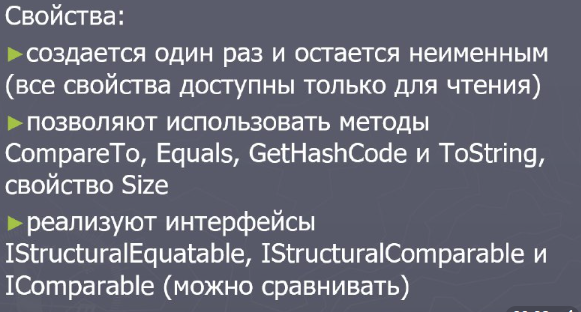
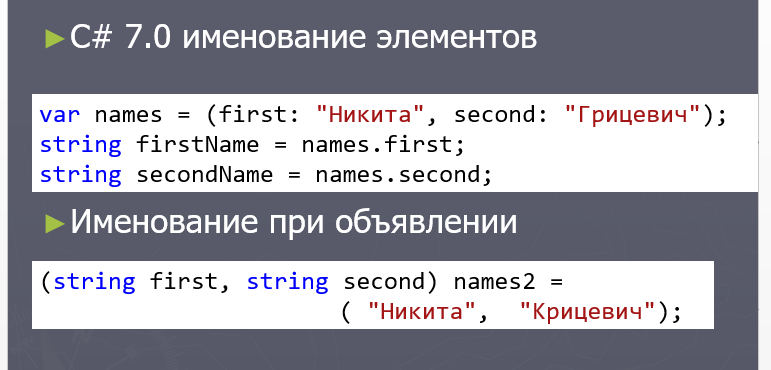


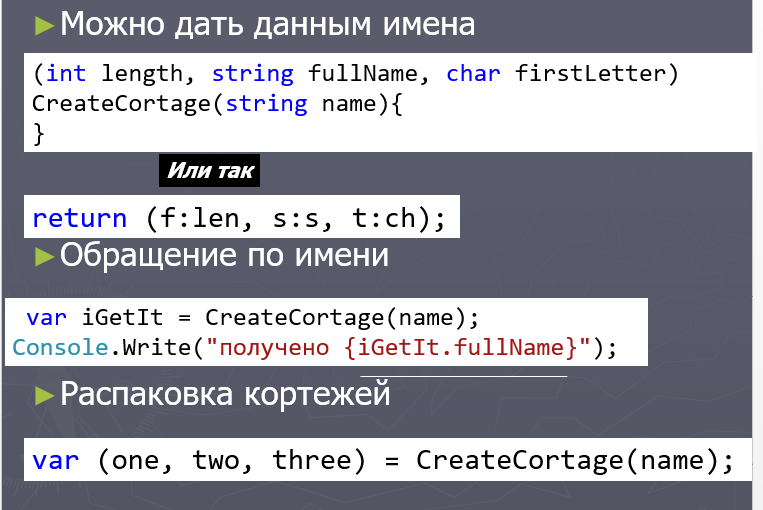


36. \*\*Кортеж\*\* - это структура данных, которая позволяет объединять несколько значений разных типов в один объект(одну группу). Он используется для возврата нескольких значений из методов или функций.

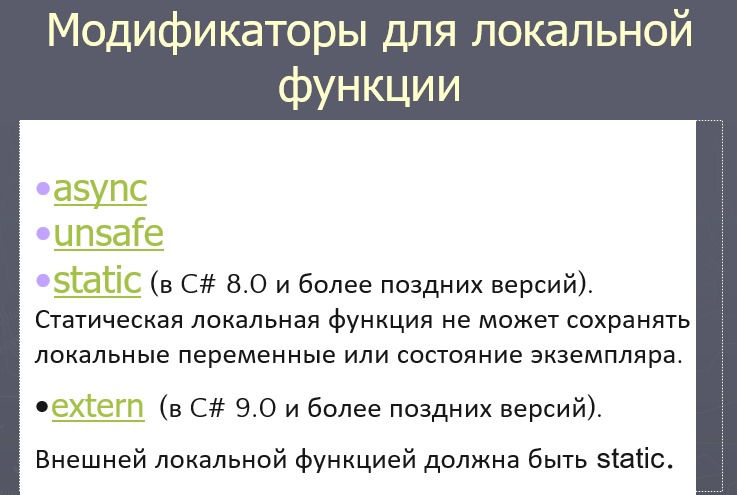
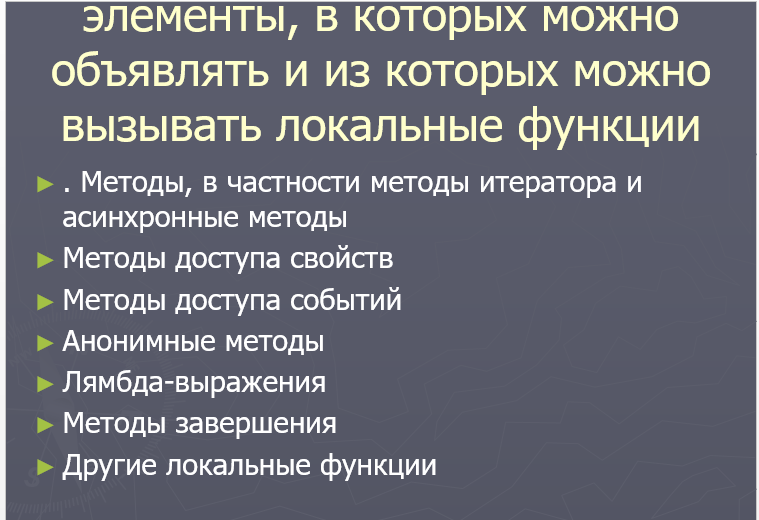
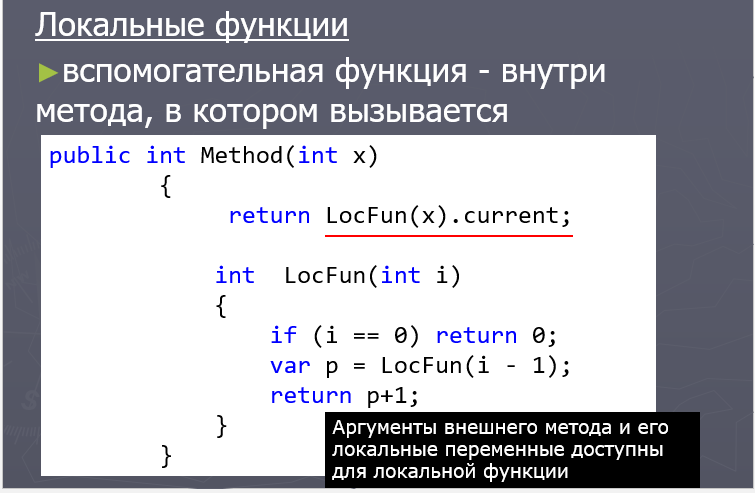
Var person = (Name: ‘John’, Age: 30);

Console.WriteLine($’Имя: {person.Name}’);





37. \*\*Локальная функция\*\* - это функция, объявленная внутри другой функции или метода. Она имеет доступ к переменным из внешней функции и может использоваться для улучшения читаемости и организации кода.



38. Код, заключенный в блок `checked`, выполняет проверку на переполнение для арифметических операций. Мб исключение. Вы можете использовать checked внутри блока кода или включить его на уровне компиляции с помощью компиляторной опции. Код, заключенный в блок `unchecked`, отключает проверку на переполнение. Это поведение по умолчанию. Однако в некоторых ситуациях, особенно при работе с битовыми операциями или когда безопасность данных критически важна, вы можете использовать эти ключевые слова, чтобы точно определить, как обрабатывать переполнения.

39. Контекст (checked/unchecked) по умолчанию определяется настройками компилятора. Его можно изменить с помощью ключевого слова `checked` или `unchecked`, а также глобальными настройками компилятора.

40. Ключевое слово `fixed` используется для создания фиксированных указателей на управляемые объекты в управляемой куче, чтобы избежать сборки мусора и управления памятью вручную. 