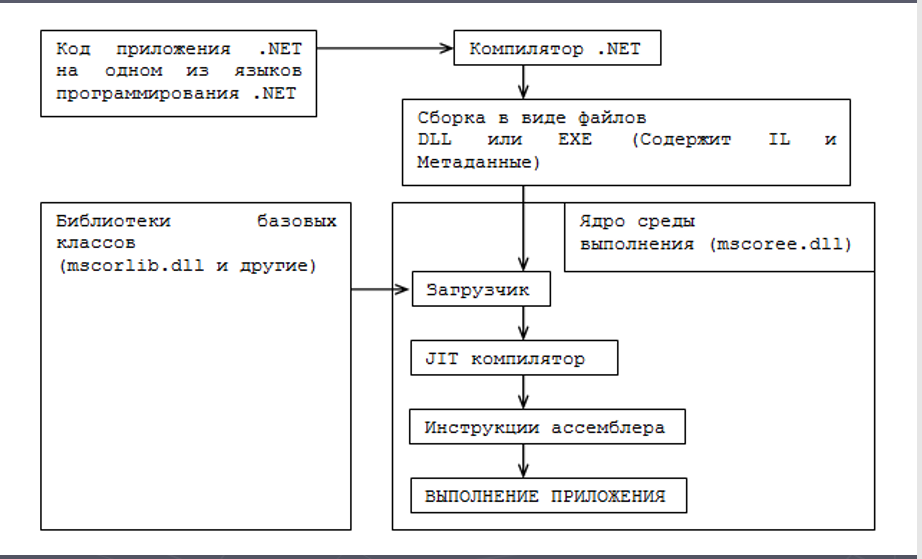
**1. Состав. NET Framework. Структура среды выполнения CLR**

**Microsoft.NET (.NET Framework) –** программная платформа. Обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows.  
**Содержит следующие основные компоненты:  
CLR (Common Language Runtime) –** общеязыковая среда исполнения, виртуальная машина на которой исполняются все приложения, работающие в среде .NET. Реализация CLI VES компанией Microsoft.  
**Компилятор JIT(Just in Time).  
MSIL(Microsoft IL) –** реализация **CLI CIL** компанией Microsoft**.   
FCL (Framework Class Library) –** реализация **CLI BCL** компанией Microsoft. Можно рассматривать, как **API CRL.**►**CLR (Common Language Runtime) –** Среда Времени Выполнения или Виртуальная Машина. Обеспечивает выполнение сборки (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация)

►**FCL (.NET Framework Class Library)** – соответствующая CLS спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений)  
Структура среды выполнения **CLR:**



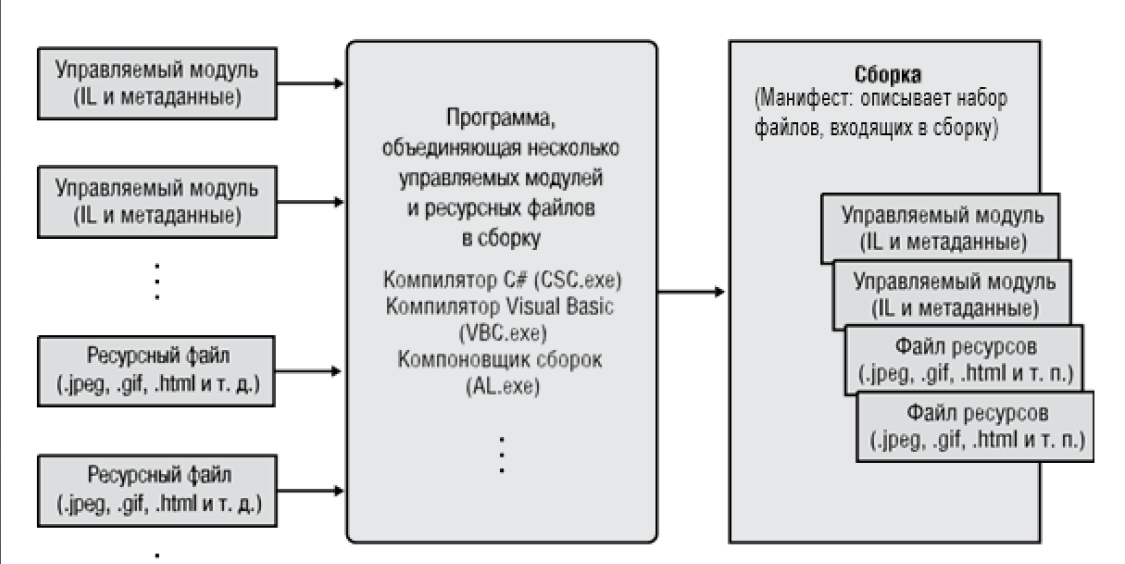
**.NET FRAMEWORK – решение следующих проблем   
1. Интеграция языков программирования.** CLS (Common Language Specification) – общеязыковая спецификация, предназначенная для разработчиков компиляторов.

CTS (Common Type Systems) - спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Microsoft выпустил несколько компиляторов, соответствующих этой спецификации: С++/CLI (C++ с управляемыми расширениями), С#, VB .NET, JScript.  **2. Работа на многих платформах.** При компиляции кода компиляторы NET Framework генерируют код на промежуточном языке (CIL, Common Intermediate Language). При исполнении CLR транслирует CIL-код в команды соответствующего процессора. В принципе, однажды .NET Framework приложение должно работать везде, где установлены и работают CLR и FCL.  
**3. Упрощенное повторное использование кода.** CLR позволяет типы разработанные на одном языке использовать в других языках.  
**4. Автоматическое управление памятью.** CLR автоматически отслеживает использование ресурсов. Сборщик мусора.  
**5. Проверка безопасности типов.** При работе в CLR практически исключена возможность записать (стереть) данные в область памяти, которая для этого не предназначена. Нет возможности передать управление в произвольную точку. **6. Единый принцип обработки сбоев.** Один из самых неприятных моментов в Windows-программирование – это отсутствие единой системы обработки ошибок и сбоев: возврат функций, коды состояний, HRESULT, исключения и т.п. Для обработки ошибок и сбоев в CLR используется только механизм исключений.

**2. Структура управляемого модуля - portable executable (PE). Понятие и исполнение сборки. CIL.**

**Portable Executable** — формат исполняемых файлов, объектного кода и динамических библиотек, используемый в 32- и 64-разрядных версиях операционной системы Microsoft Windows. Формат PE представляет собой структуру данных, содержащую всю информацию, необходимую PE-загрузчику для отображения файла в память. Исполняемый код включает в себя ссылки для связывания динамически загружаемых библиотек, таблицы экспорта и импорта API функций, данные для управления ресурсами и данные локальной памяти потока (TLS).

**Сборка (assembly) —**1) это абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов.   
2) дискретная единица многократно используемого кода внутри CLR.



**Исполнение сборки  
JIT-компилятор (Just-In-Time)**1) CLR ищет типы данных и загружает во внутренние структуры   
2) Для каждого метода CLR заносит адрес внутренней CLR функции JITCompiler   
3) JITCompiler ищет в метаданных соответствующей сборки IL-код вызываемого метода, проверяет и компилирует IL-код в машинные команды   
4) Они хранятся в динамически выделенном блоке памяти.   
5) JITCompiler заменяет адрес вызываемого метода адресом блока памяти, содержащего готовые машинные команды   
6) JITCompiler передает управление коду в этом блоке памяти.

**Common Intermediate Language** (сокращённо **CIL**) — «высокоуровневый ассемблер» виртуальной машины .NET. Промежуточный язык для платформы .NET Framework. JIT-компилятор CIL является частью CLR.  
Все компиляторы, поддерживающие платформу .NET, должны транслировать код с языков высокого уровня платформы .NET на язык CIL. В частности, код на языке CIL генерируют все компиляторы .NET фирмы «Microsoft», входящие в среду разработки Microsoft Visual Studio.  
По синтаксису и мнемонике язык CIL напоминает язык ассемблера. Его можно рассматривать как ассемблер виртуальной машины .NET. В то же время язык CIL содержит некоторые достаточно высокоуровневые конструкции, повышающие его уровень по сравнению с ассемблером для любой реально существующей машины, и писать код непосредственно на CIL легче, чем на ассемблере для реальных машин. Поэтому CIL можно рассматривать как своеобразный «высокоуровневый ассемблер».   
Язык CIL также нередко называют просто IL— буквально «промежуточный язык».   
Синтаксис и мнемоника языка CIL описываются стандартом «ECMA-335». Спецификация CIL является составной частью более общей спецификации — спецификации CLI (***c****ommon* ***l****anguage* ***i****nfrastructure*).

**3. CTS (Common Type System). Типы данных C#. Ссылочные и типы значений**

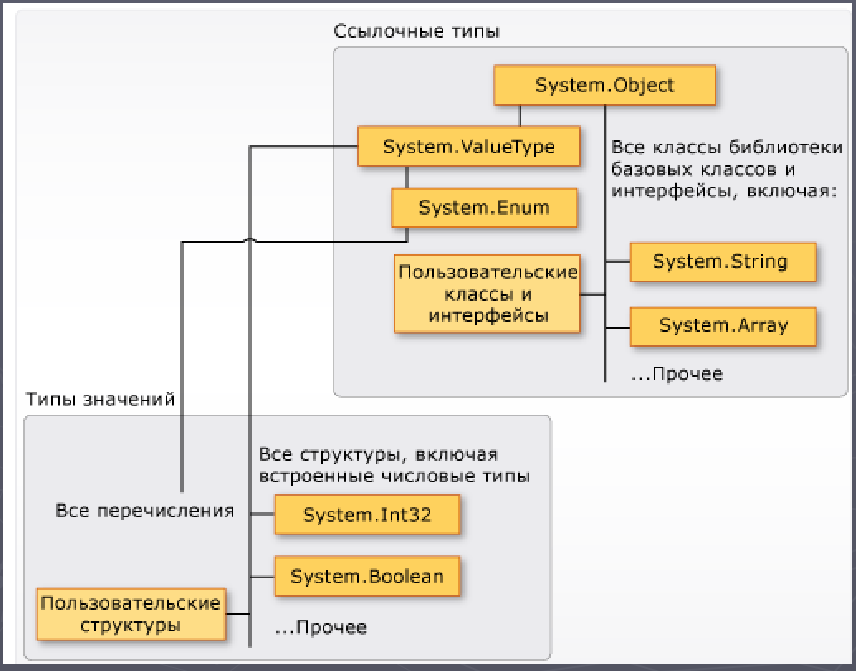
**CTS (Common Type Systems)-** спецификацию типов, которые должны поддерживаться всеми языками ориентированными на CLR. Microsoft выпустил несколько компиляторов соответствующих этой спецификации: С++/CLI (C++ с управляемыми расширениями), С#, VB .NET, Jscript.   
**Типы данных C#** поддерживает общую систему типов (CTS): для объявления того или иного встроенного типа данных из CTS обычно предусмотрено свое уникальное ключевое слов.  
**Типы значений (value types)** – хранятся в стеке

1) Определяются struct или emun

2) Размещение в стеке потока

3) поля экземпляра размещаются в самой переменной

4) не обрабатываются cборщиком мусора. Типы значения, структуры, перечисления, простые типы.  
**Ссылочные типы (reference types).** определяются class (в куче)



**Где выделяется память под ссылочные типы?**

1)обычно в стеке потока, но могут быть встроены в ссылочные типы

2)память выделяется из очереди

3)память выделяется из управляемой кучи

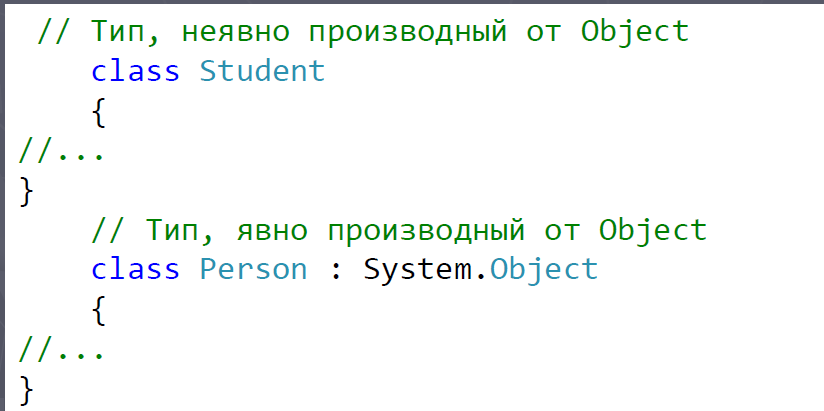
4)память выделяется из кэша

По умолчанию: value-type -

Типы данных, которые поддерживаются компилятором напрямую, называются **примитивными (primitivetypes**) или встроенными у них существуют прямые аналоги в библиотеке классов .NET FrameworkClassLibrary

**Встроенные (примитивные) типы C#.**Тип C# - Размер в битах - Тип System (FCL).   
sbyte - 8 - System.Sbyte;  
short - 16 - System.Int16;  
int – 32 - System.Int32;   
long – 64 - System.Int64;   
byte – 8 - System.Byte;  
ushort – 16 - System.Uint16;   
uint - 32 - System.Uint32;   
ulong – 64 - System.Uint64;   
char - 16 - System.Char;   
bool – 8 - System.Boolean;   
float - 32 - System.Single;  
double - 64 - System.Double;  
decimal – 128 - System.Decimal;   
string - System.String;  
object - System.Object;

►В CLR каждый объект прямо или косвенно является производным от System.Object

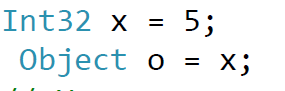


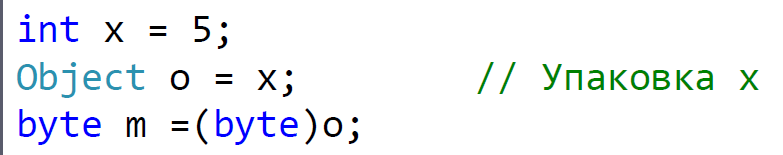
**Значимый тип может хранить значение по умолчанию – “ “ или 0, а ссылочный – NULL.**

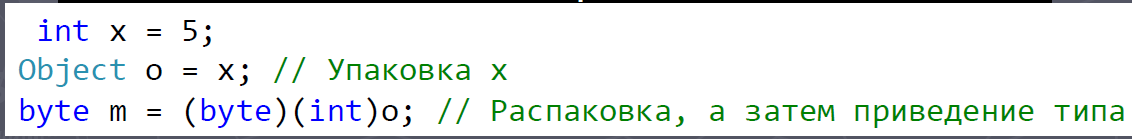
**4. Понятие упаковки и распаковки типов. Типы Nullable: преобразование, проверка, null-объединение**

**Упаковкой (boxing)** называется процесс преобразования типа значения в тип System.Object или в тип интерфейса, который реализуется данным типом значением  
1. в управляемой куче выделяется память

2. поля копируются

3. возвращается адрес объекта

**Распаковка (unboxing) -** получение указателя на исходный значимый тип (поля данных), содержащийся в объекте  
Объекты значимого типа существуют в двух формах: неупакованной (unboxed) и упакованной (boxed). Ссылочные типы бывают только в упакованной форме.  


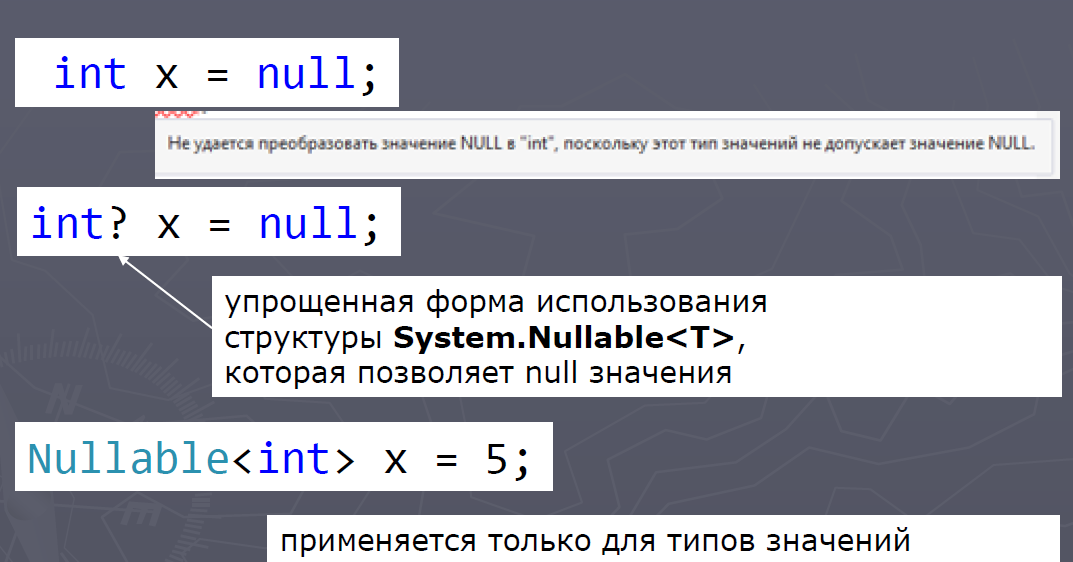
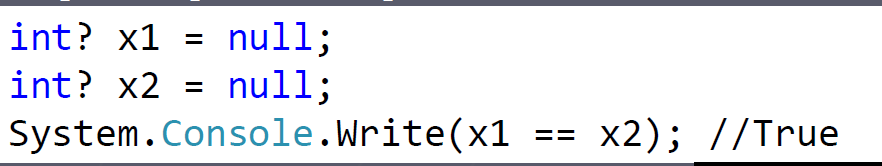
1. Если переменная, содержащая ссылку на упакованный значимый тип, равна null, генерируется исключение NullReferenceException.  
2. Если ссылка указывает на объект, не являющийся упакованным значением требуемого значимого типа, генерируется исключение InvalidCastException  


Назначение:

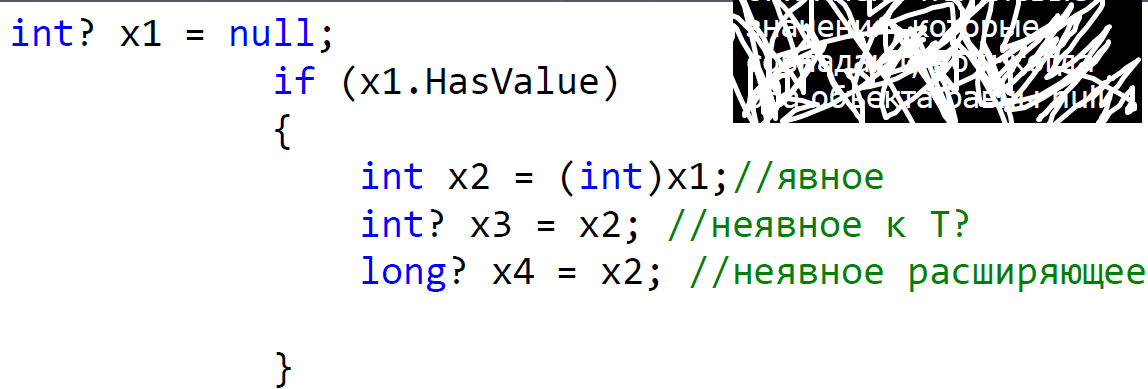
►позволяет использовать типы-значения в коллекциях ( где элементы являются элементами типа object)

►внутренний механизм, который обеспечивает возможность вызывать для типов-значений, подобных int и struct, методы Object. **упаковка и распаковка/копирование снижают производительность приложения!**

**Nullable:**

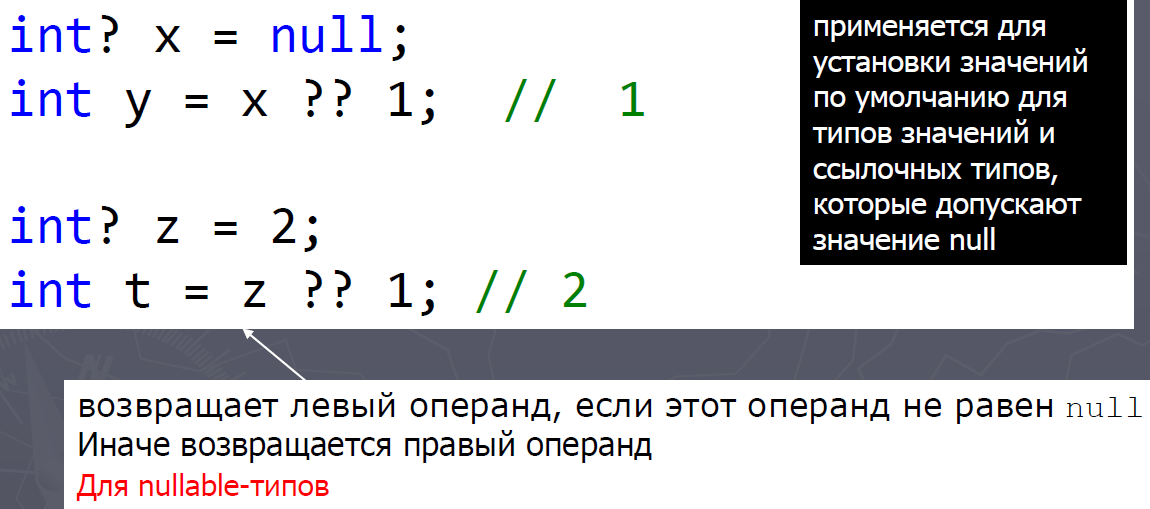
 **Проверка на равенство объектов**

Равны не только, когда они имеют ненулевые значения, которые совпадают, но и когда оба объекта равны null

**Преобразование****

*Чтобы получить доступ к значению Nullable, надо использовать свойство* ***Value***

**Оператор ?? (null-объединение)**



**5. Тип данных String: операции, литералы, пустые и нулевые строки, форматированный вывод.**

**Тип string** предназначен для работы со строками символов в кодировке Unicode. Ему соответствует базовый класс System.String библиотеки .NET.  
**Создание строки:**

**Операции для строк**

► присваивание (=);

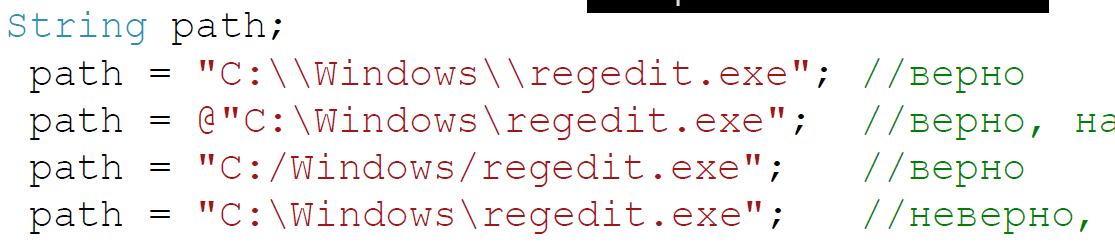
► проверка на равенство содержимого (==);

► проверка на неравенство (!=);

► обращение по индексу ([]);

► сцепление (конкатенация) строк (+)

► <,>, >=,<= - сравнивают ссылки!!!!!!!!

**Строки** равны, если имеют одинаковое количество символов и совпадают посимвольно.  
**Обращаться** к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения.  
**Строки** типа string относятся к неизменяемым типам данных.  
**Методы**, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.  
**Строковые литералы**

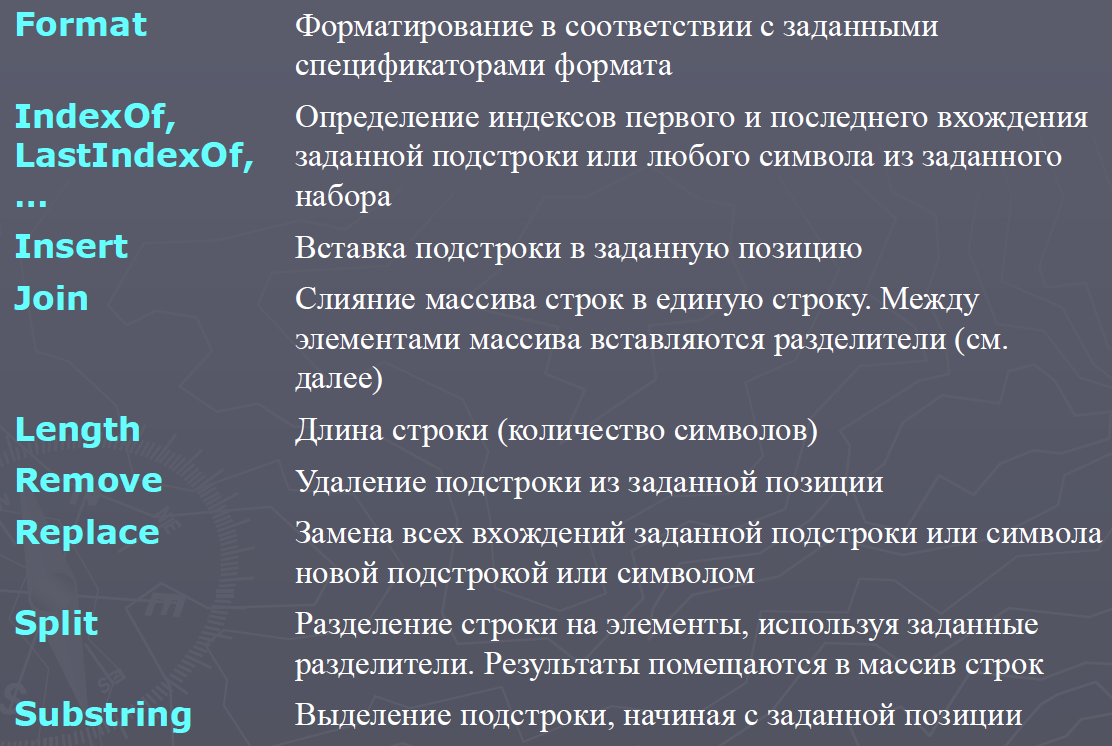
►Поставить ещё один символ обратной косой черты (“\\”);

►Предварить строковый литерал символом ‘@’;

►В случае, если литерал есть путь к файлу (локальный или сетевой), каталогу или web-странице, можно использовать альтернативный символ '/' вместо “\\”.

**Некоторые элементы класса System.String**

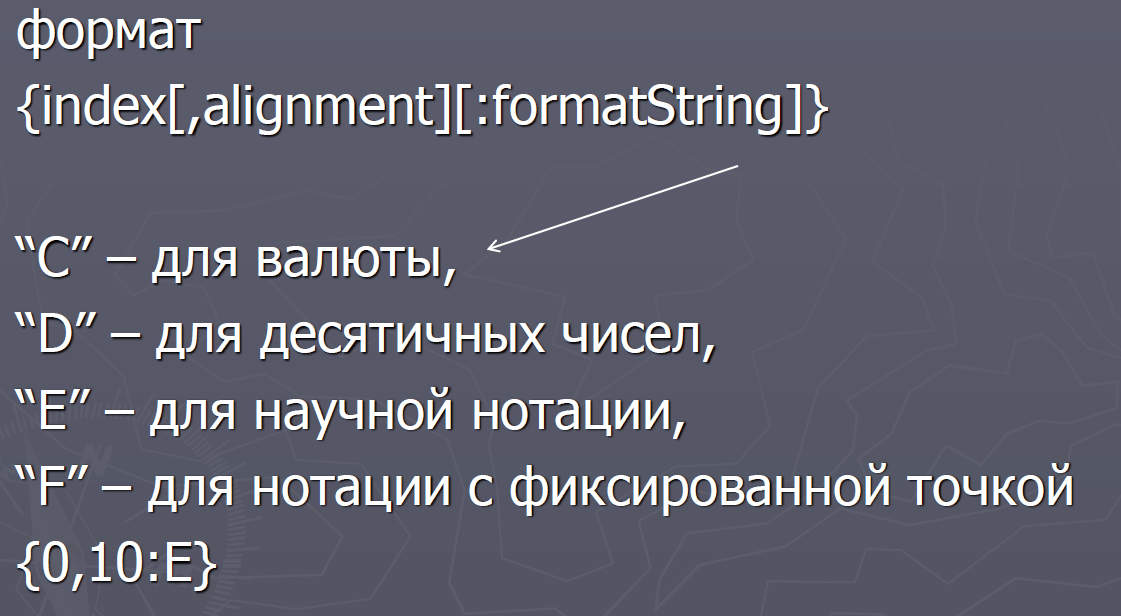
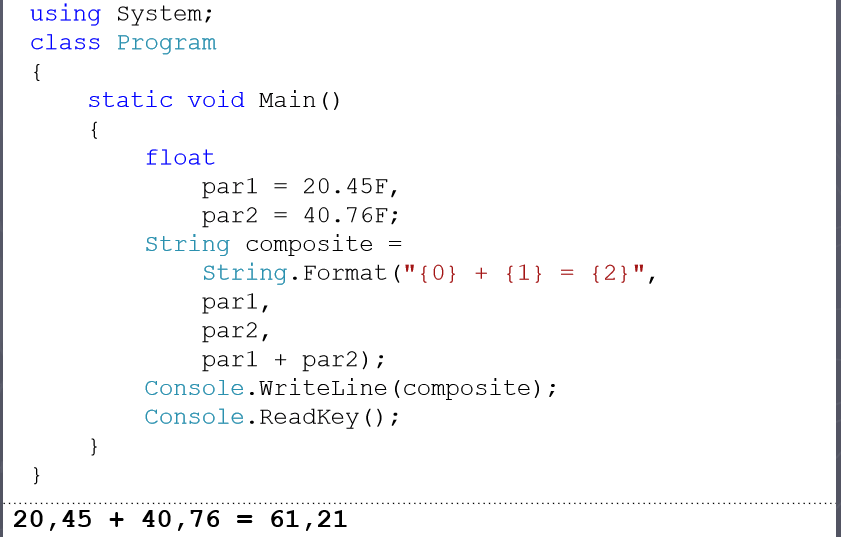




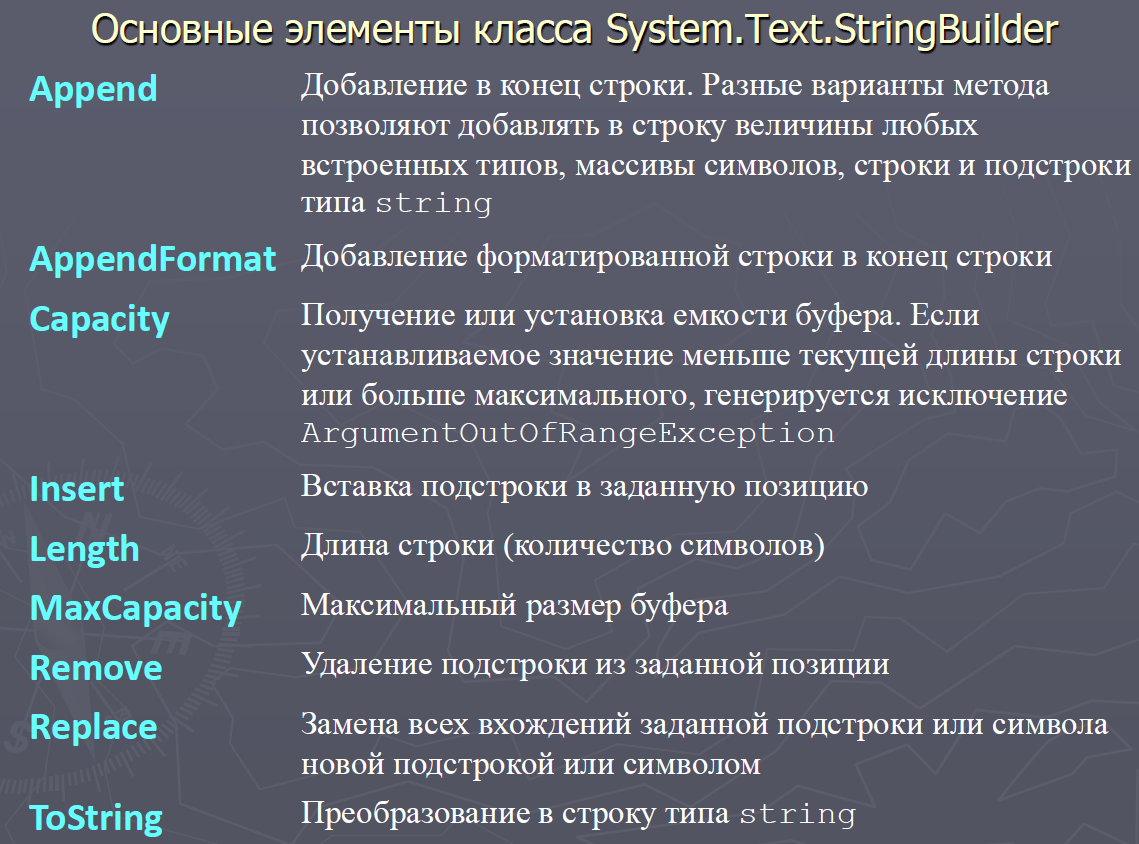
**Пустые строки и строки null**

**Пустая строка —** экземпляр объекта System.String, содержащий 0 символов:  
**string s = "";**  
Для пустых строк можно вызывать методы.

**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.

**Форматированный вывод. Метод String.Format**  


**Класс StringBuilder – по сравнению с классом string, может изменять значение строки (в классе string происходит вначале копия строки, а затем уже изменяется (а старая - удаляется)). Можно сказать, это динамическая строка.**



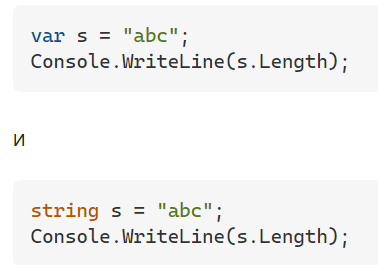
**6. Неявная типизация – назначение и использование.**

**Неявная типизация –** это возможность явным образом не указывать тип данных при инициализации переменной. В С# есть такая возможность: при помощи типа данных dynamic и var.

**Основное** **различие** **между** **var** **и** динамическими ключевыми словами **состоит** в том, что время привязки отличается: **var** - раннее связывание, динамическое связывание будет выполняться во время выполнения. **var** на самом деле является синтаксическим сахаром, выдаваемым нам компилятором.

Еще одно отличие заключается в том, что когда мы имеем дело с ключевым словом dynamic, мы не получим никаких ошибок во время компиляции.

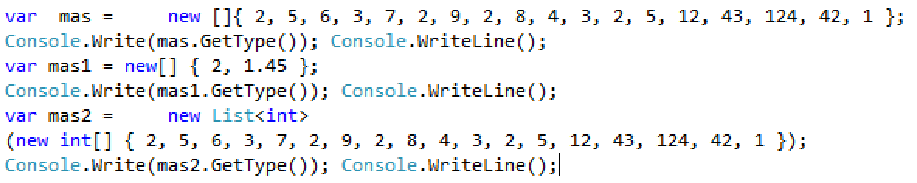
**Это одно и тоже(var):**



**Dynamic выполняется во время выполнения, т.е. это будет правильно:**



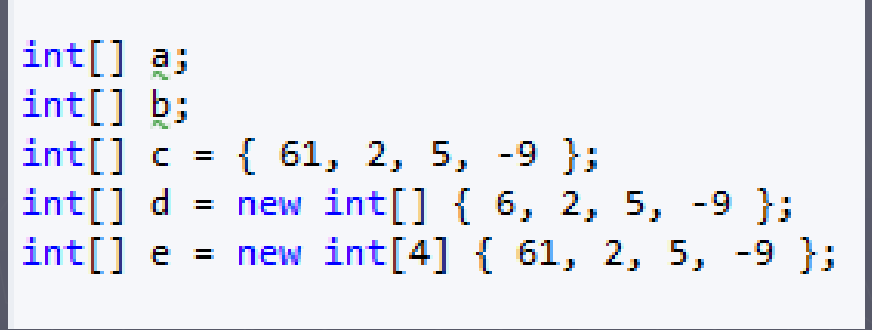
**Ограничение динамических типов**-могут использоваться только для обращения к членам экземпляров ( должна ссылаться на объект)  
- не могут использовать лямбда-выражения или анонимные методы C# при вызове метода   
- не могут воспринимать расширяющие методы

**Неявно типизированные локальные переменные. Ключевое слово var** **При компиляции компилятор сам выводит тип данных исходя из присвоенного значения  
Ограничения:   
►**должно включать инициализатор (null нельзя)  
►ключевое слово 'var' может применяться только в объявлении локальной переменной   
►неявно типизированные локальные переменные не допускают множественного объявления

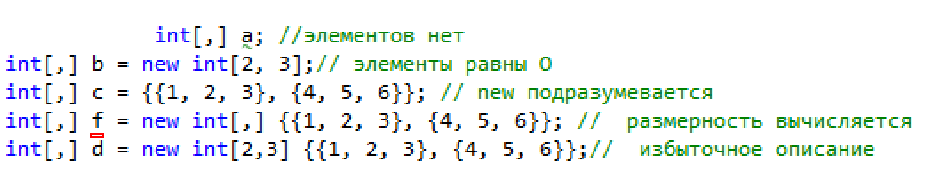
**7. Массивы C# одномерные, прямоугольные и ступенчатые.**

**Массив** представляет набор однотипных данных. Объявление массива похоже на объявление переменной за тем исключением, что после указания типа ставятся квадратные скобки.

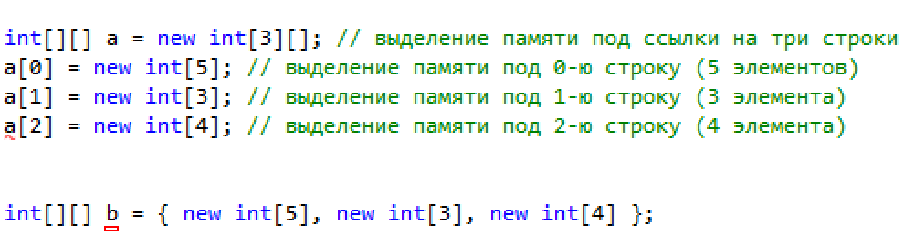
**Одномерные массивы (single-dimensional):**



**Прямоугольные (многомерные) массивы (матрица):**

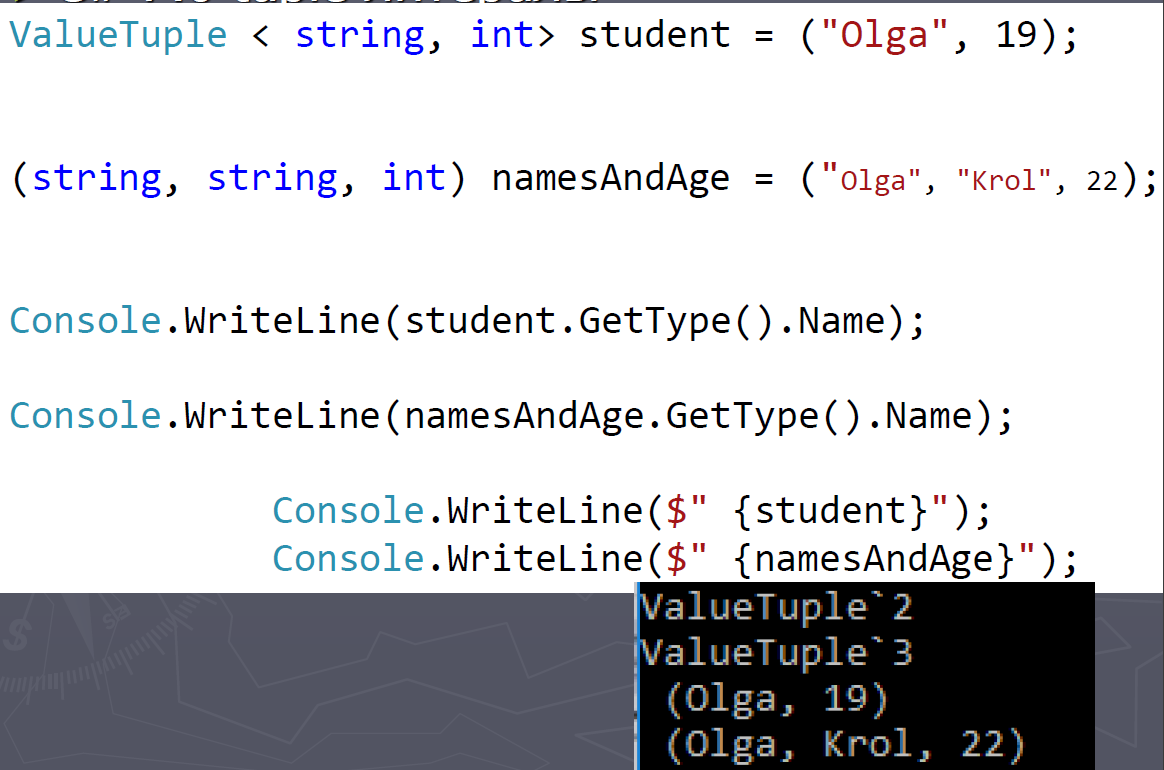


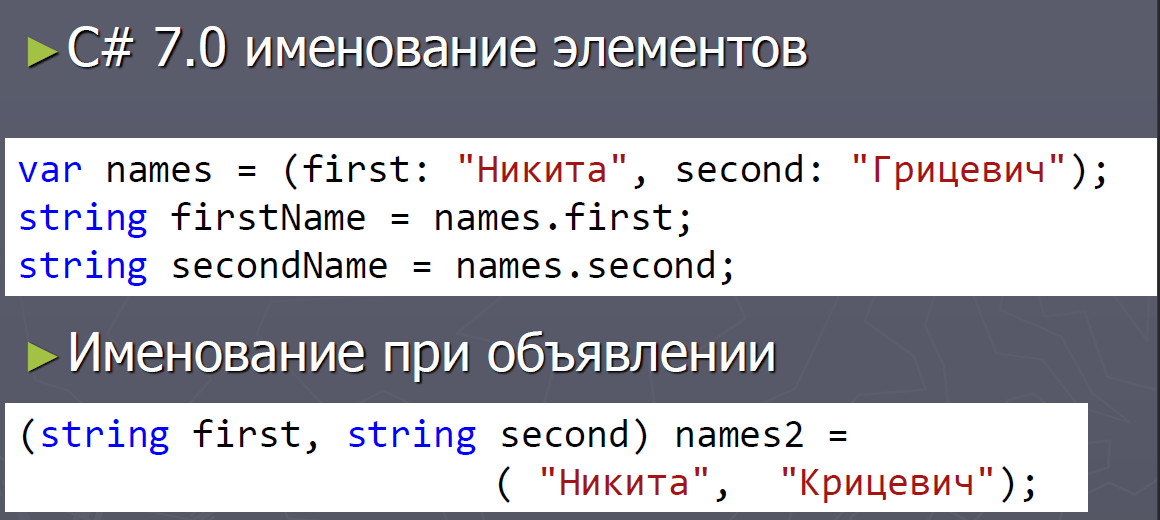
**Ступенчатые, зубчатые массивы (**это массив массивов, в котором длина каждого массива разная**):**

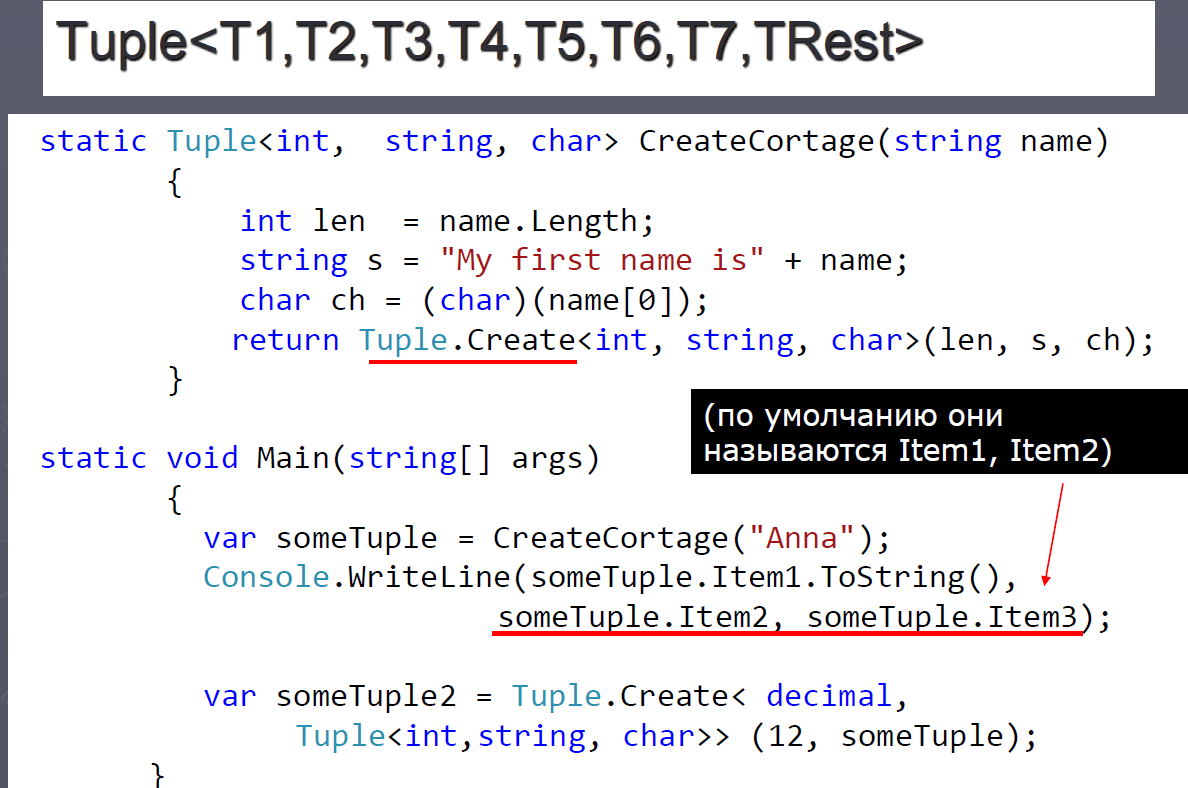


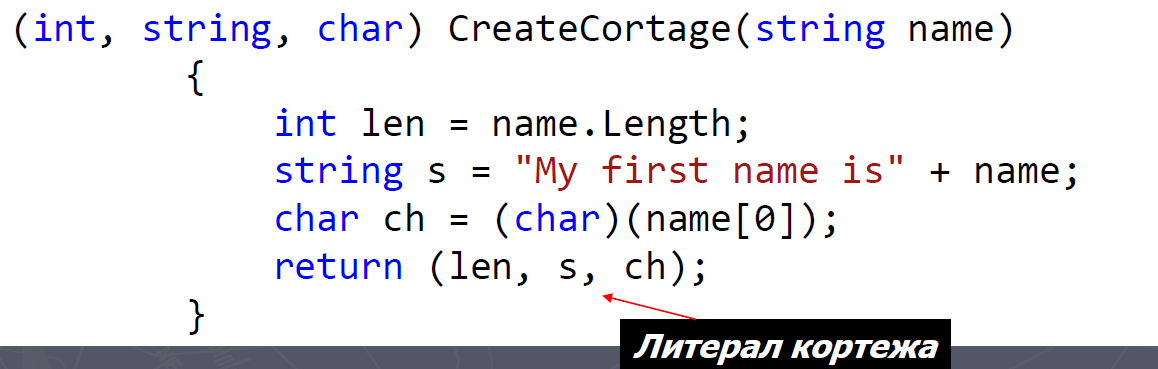
**8. Понятие кортежей. Свойства, создание**

**Кортежи (tuple)** комбинируют объекты различных типов (от одного до восьми).



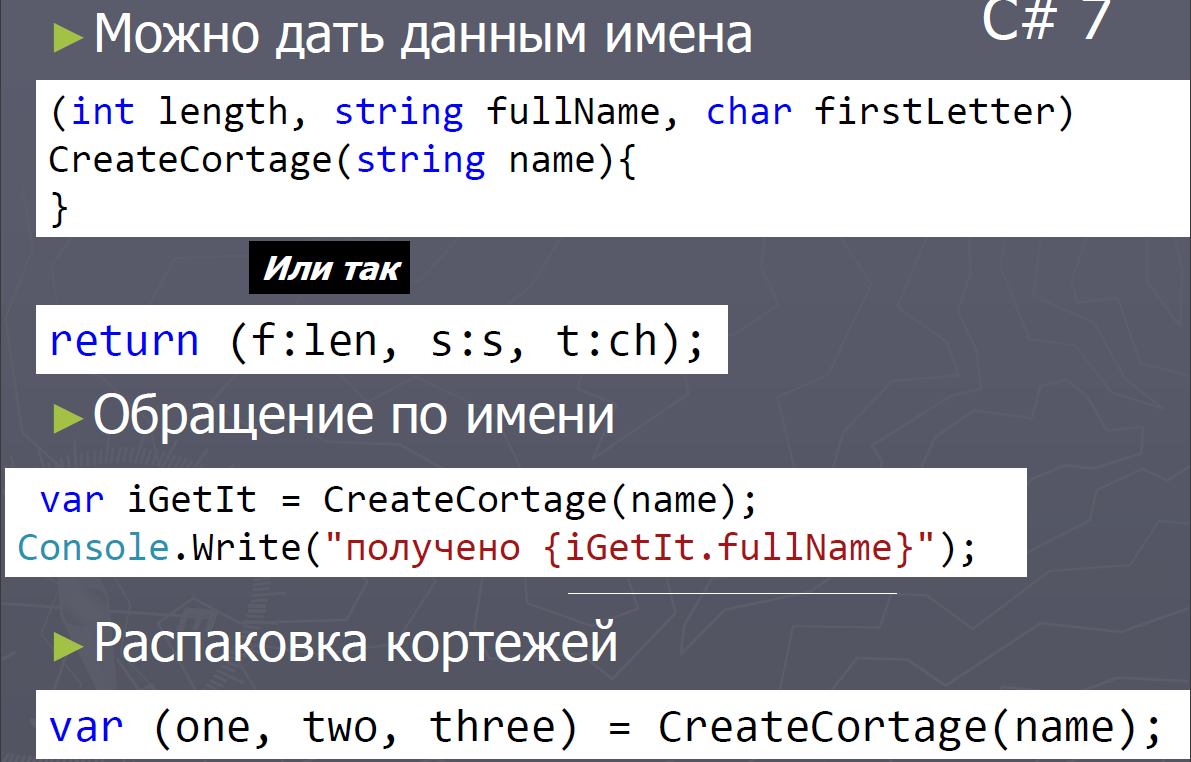




**Свойства:**►создается один раз и остается неименным (все свойства доступны только для чтения)   
►позволяют использовать методы CompareTo, Equals, GetHashCode и ToString, свойство Size   
►реализуют интерфейсы IStructuralEquatable, IStructuralComparable и IComparable (можно сравнивать)  


Кортежи -значимый тип, System.ValueTuple.

Элементы -открытые поля, могут сравниваться на равенство, присвоится др.др



**9. Принципы объектно-ориентированного программирования.**

**Объектно-ориентированное программирование —** это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

**Принципы объектно-ориентированного программирования**

**►Наследование (Inheritance) -**1) процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта.  
2) иерархии классов. Student -> GradStudent  
  
**►Инкапсуляция (Encapsulation) -** механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования1) Никто не знает что внутри

2) Никто не может менять данные снаружи  
Свойства инкапсуляции

►Совместное хранение данных и функций

►Сокрытие внутренней информации от пользователя

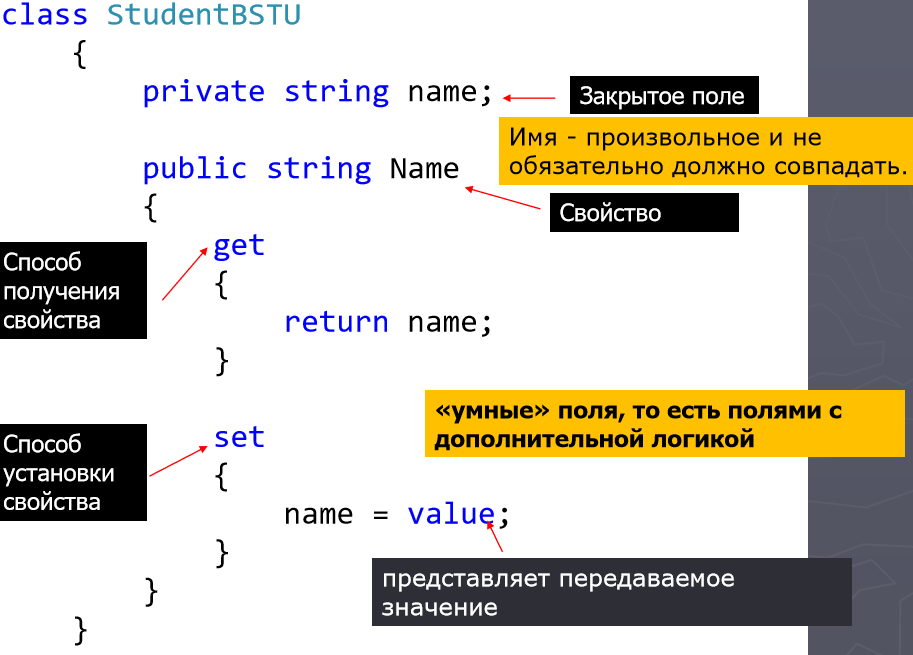
►Изоляция пользователя от особенностей реализации.  
  
**►Полиморфизм (Polymorphism).**1) Способность вызывать метод потомка через экземпляр предка  
2) Поддержка полиморфизма осуществляется через виртуальные функции, механизм перегрузки функций и операторов, а также обобщения  
  
**►Абстракция данных**. Подразумевает разделение и независимое рассмотрение интерфейса и реализации. Абстракция - уровень описания/представления модели чего - либо.

**10. Класс. Элементы класса. Свойства и индексаторы.**

► Класс – это некоторое абстрактное понятие - шаблон, по которому определяется форма объекта

► Объект – это физическая реализация класса(шаблона).



**Свойства** – специальные методы класса, служат для организации доступа к полям класса. Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки (предоставляет инкапсуляцию).  


Назначение - свойства позволяют вложить дополнительную логику  
Ограничения свойств:

1) не может быть передано методу в качестве параметра ref или out. 2) не подлежит перегрузке

3) не должно изменять состояние базовой переменной при вызове аксессора get

4) могут быть статическими, экземплярными, абстрактными и виртуальными

5) могут иметь модификатор доступа

6) могут определяться в интерфейсах  
Автоматические свойства: тип имя { get; set; }. Компилятор автоматически реализует методы для правильного возвращения значения из поля и назначения значения полю  
Проблемы:

• неявная инициализация

• проблемы при сериализации и десериализации

• во время отладки нельзя установить точку останова

**Сокращенное объявление:**

***тип имя{ get; set; }***

компилятор автоматически реализует методы для правильного возвращения значения из поля и назначения значения полю

Проблемы:

•неявная инициализация

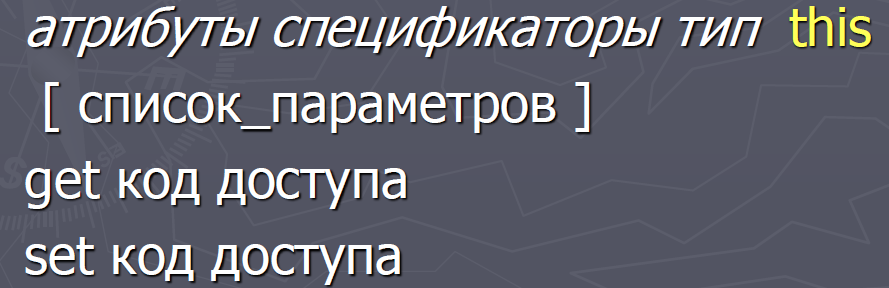
•проблемы при сериализациии десериализации

•во время отладки нельзя установить точку останова

**Индексаторы (свойства с параметрами)**

►Позволяют индексировать объекты таким же способом, как массив или коллекцию ►«умный» индекс для объектов

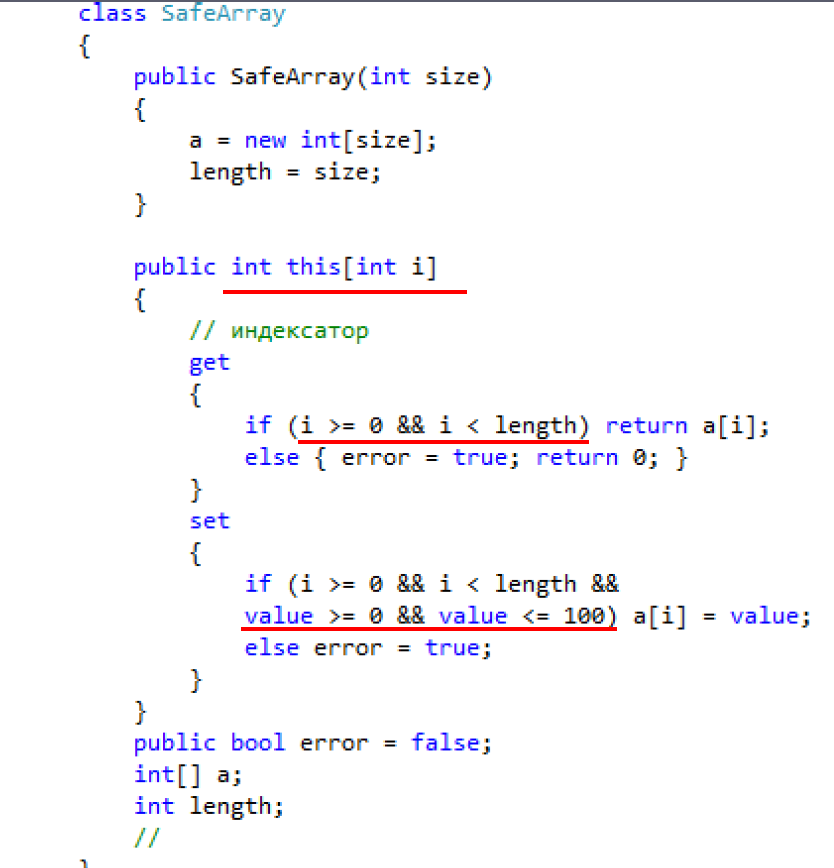
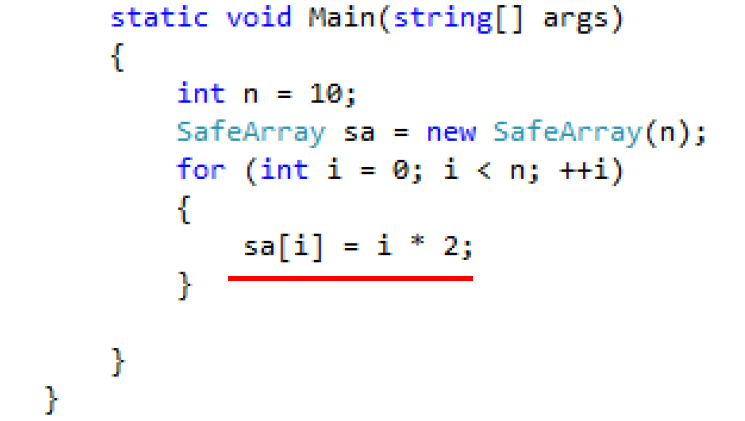
►средство, позволяющее разработчику перегружать оператор []



**Ограничения на индексаторы:**

1) значение, выдаваемое индексатором, нельзя передавать методу в качестве параметра ref или out

2) индексатор не может быть объявлен как static. Можно перегружать

**11. Класс. Константы. Поля только для чтения. Инициализаторы класса.**

► Класс – это некоторое абстрактное понятие - шаблон, по которому определяется форма объекта  
► Объект – это физическая реализация класса(шаблона).

**Константы**  
**const int CC =100;** // значение не может изменено   
1) компилятор сохраняет значение константы в метаданных модуля

 константы можно определять только для таких типов, которые компилятор считает примитивными (или не примитивный но тогда = null)

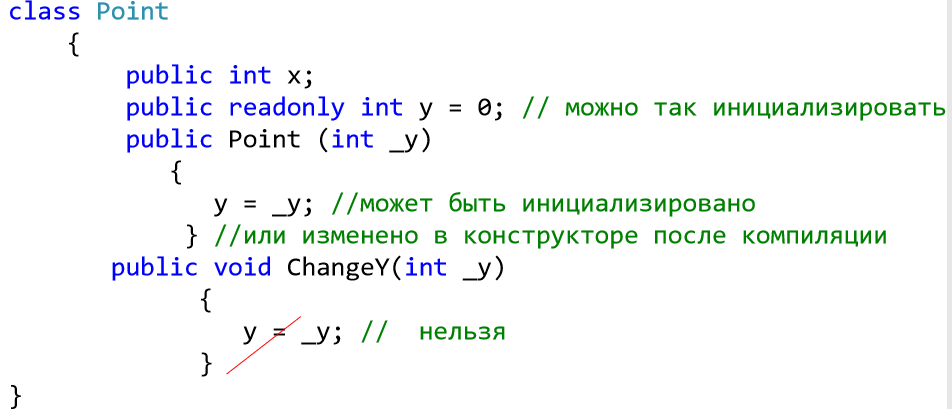
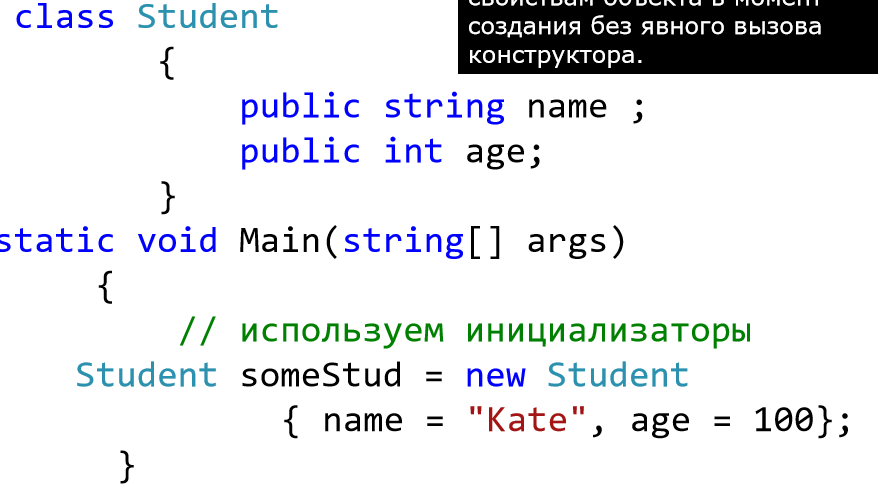
2) константы считаются не явно статическими, всегда связаны с типом, а не с экземпляром тип

3) нельзя получать адрес константы и передавать ее по ссылке

4) определять можем один раз

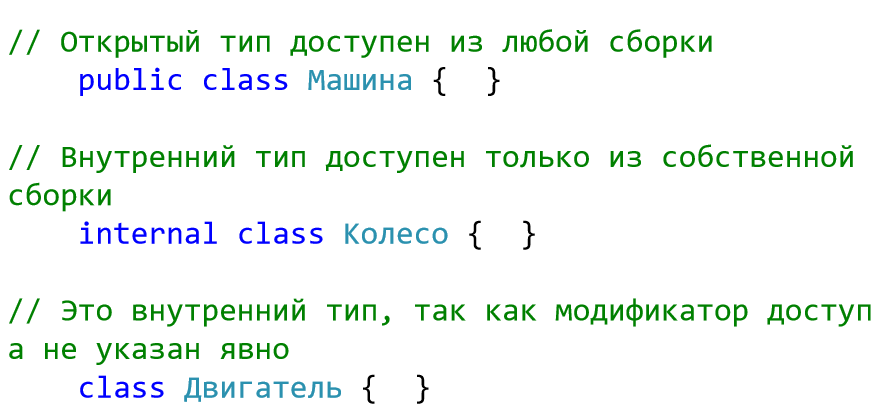
5) к моменту компиляции они должны быть определены.  
**Поля для чтения readonly**

1) Запись в поле разрешается при объявлении или в коде конструктора

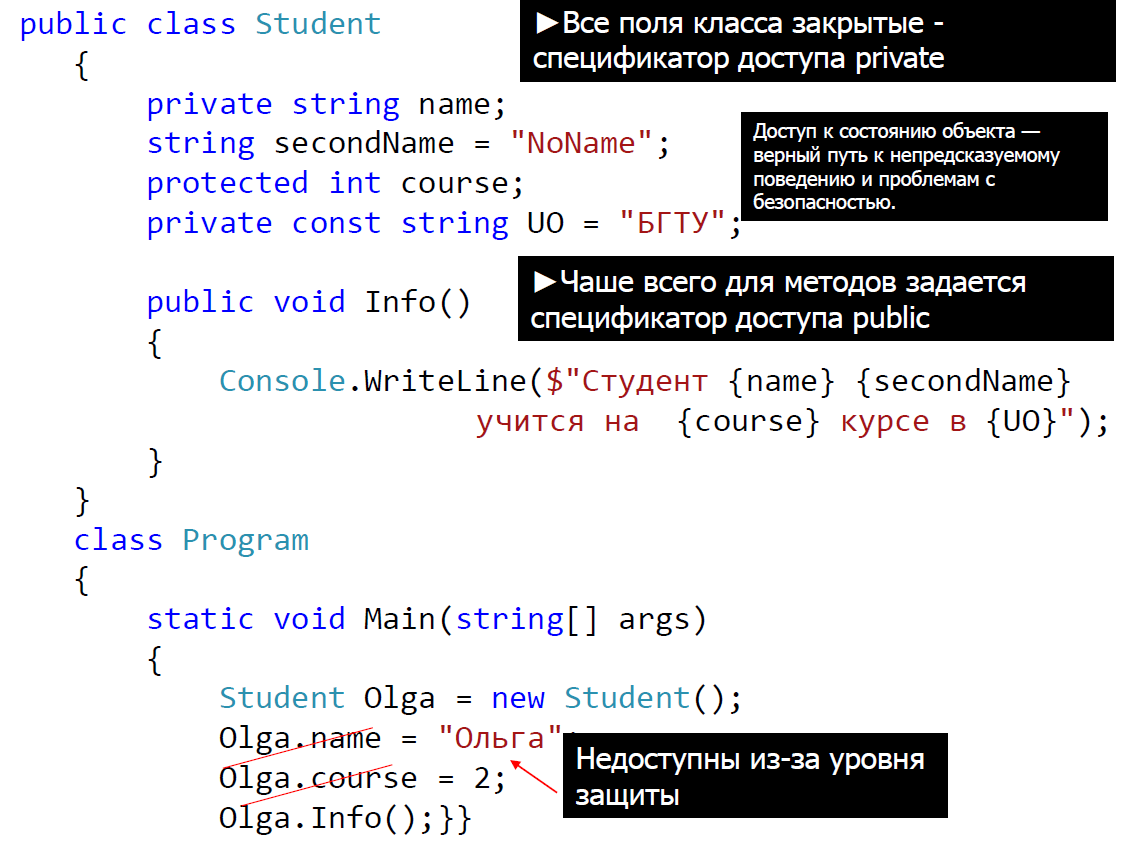
2) Инициализировать или изменять их значение в других местах нельзя, можно только считывать их значение.  
  
**Инициализаторы**С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.  


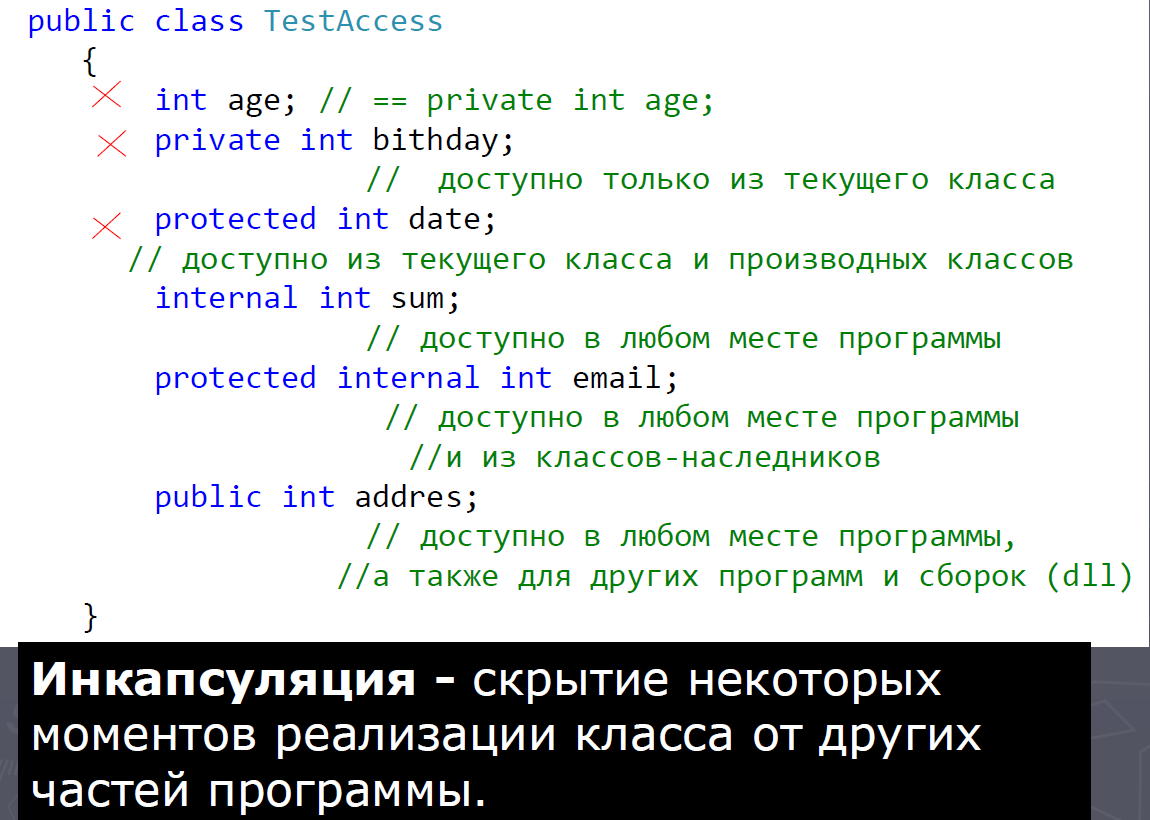
**12. Спецификаторы доступа C#. Видимость типов. Доступ к членам типов.**

**Видимость типа** ► может быть открытым (public) или внутренним (internal).



**Доступ к членам типов   
►public -** доступ не ограничен – все члены во всех сборках **►private -** по умолчанию для членов класса (используется для вложенных классов). Доступен только методам в определяющем типе и вложенных в него типах **►protected -** (используется для вложенных классов) Доступен только методам в определяющем типе (и вложенных в него типах) или в одном из его производных типов независимо от сборки **► internal -** доступ только из данной сборки





**13. Класс. Конструкторы и их свойства. Деструкторы.**

► Класс – это некоторое абстрактное понятие - шаблон, по которому определяется форма объекта  
► Объект – это физическая реализация класса(шаблона). **Конструкторы —** это специальные методы, позволяющие корректно инициализировать новый экземпляр типа.

**Создание экземпляра объекта ссылочного типа**

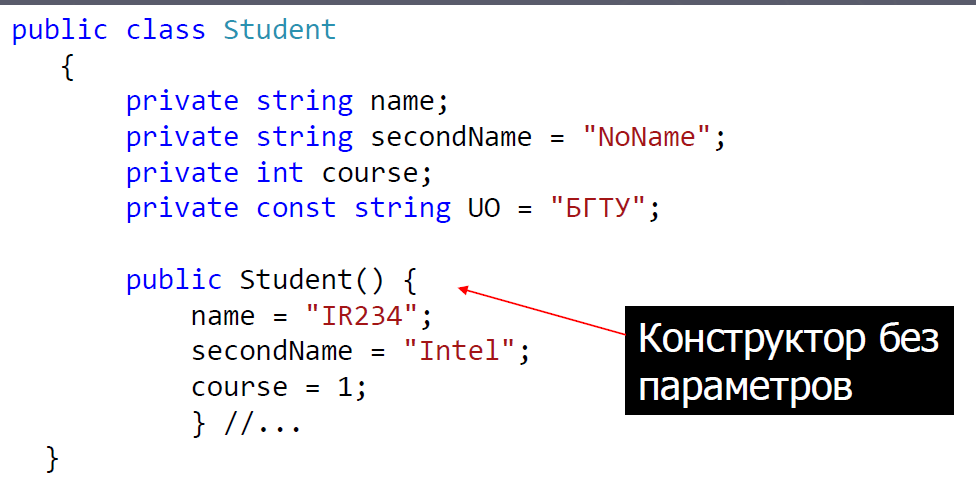
1) выделяется память для полей данных экземпляра

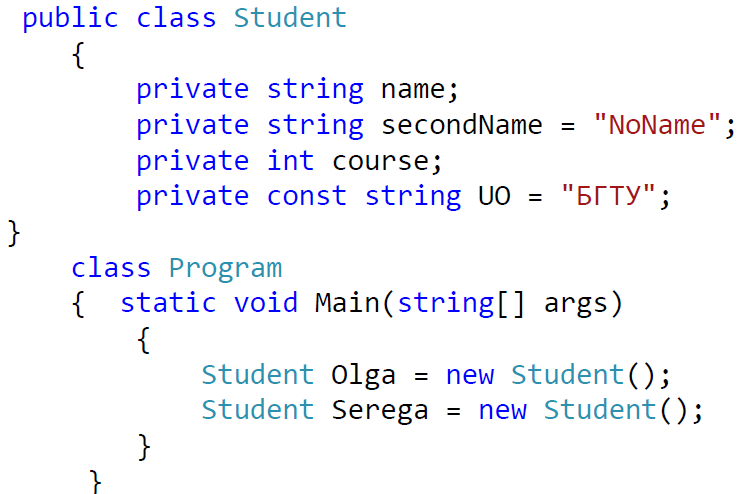
2) инициализируются служебные поля

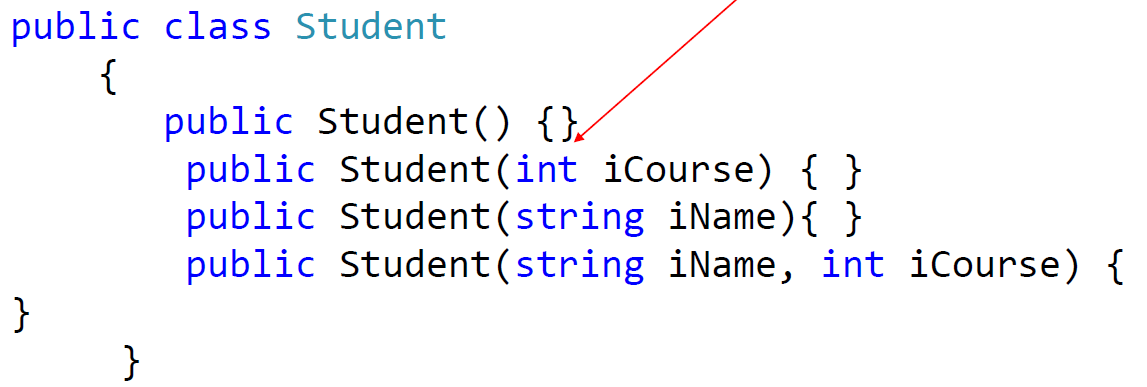
3) вызывается конструктор экземпляра, устанавливающий исходное состояние нового **Объекта память всегда обнуляется до вызова конструктора экземпляра типа.**

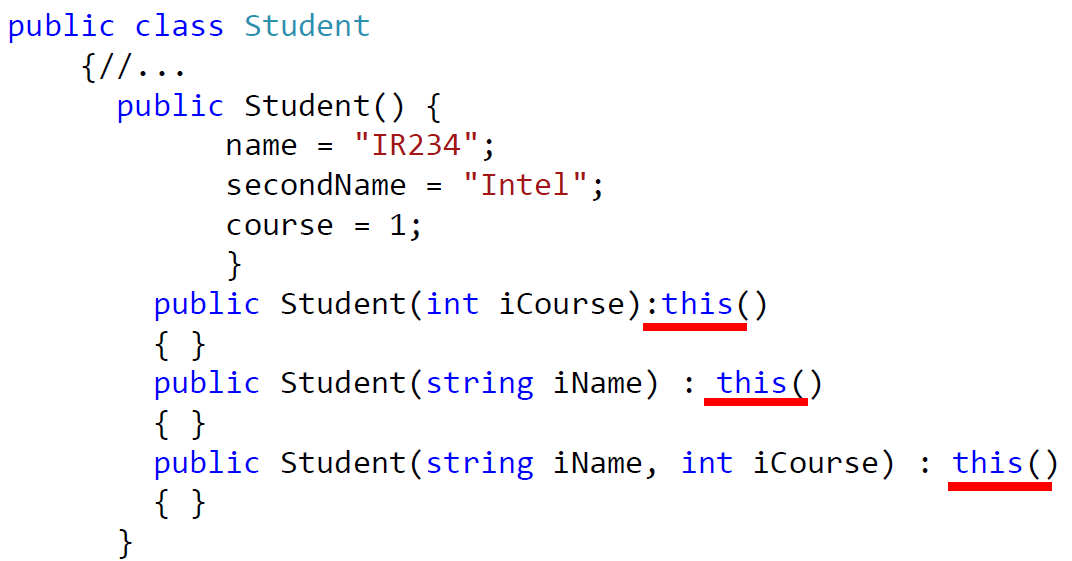
Любые поля, не задаваемые конструктором явно, гарантированно содержат 0 или null.

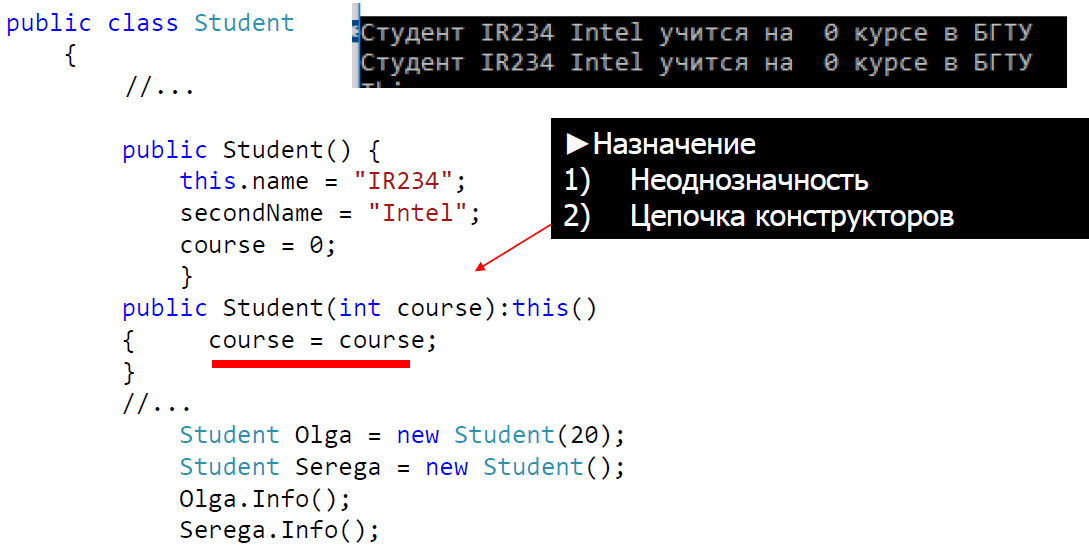
**Свойства конструкторов**   
►1) не имеет возвращаемого значения   
►2) имя такое же как и имя типа (класса)

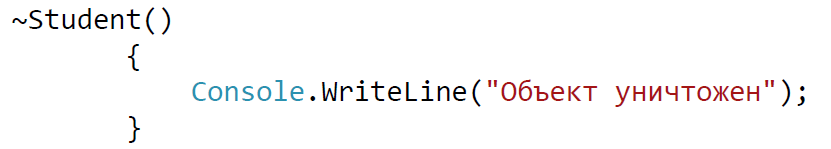
  
► 3) не наследуются   
► 4) нельзя применять модификаторы virtual, new, override, sealed и abstract   
► 5) для класса без явно заданных конструкторов компилятор создает конструктор по умолчанию (без параметров)

  
► 6) для статических классов (запечатанных и абстрактных) компилятор не создает конструктор по умолчанию   
► 7) может определяться несколько конструкторов, сигнатуры и уровни доступа к конструкторам обязательно должны отличаться

  
► 8) можно явно заставлять один конструктор вызывать другой конструктор посредством зарезервированного слова this: обеспечивает доступ к текущему экземпляру класса



► в любой нестатический метод автоматически передается скрытый параметр this  
 

**Деструкторы**  
► вызываться непосредственно перед окончательным уничтожением объекта системой "сборки мусора", чтобы гарантировать четкое окончание срока действия объекта.  
**~имя\_класса () { // код деструктора }**   


**Нельзя узнать, когда именно вызовется деструктор**

**Если программа завершиться до того, как произойдет "сборка мусора", деструктор может быть вообще не вызван**

**Свойства деструктора**  
►Класс может иметь только один деструктор.

►Деструкторы не могут быть унаследованы или перегружены.

►Деструкторы невозможно вызвать. Они запускаются автоматически.

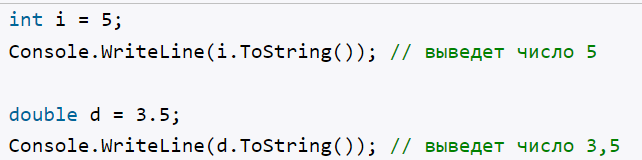
►Деструктор не принимает модификаторы и не имеет параметров.

**14. Класс и методы System.Object.**

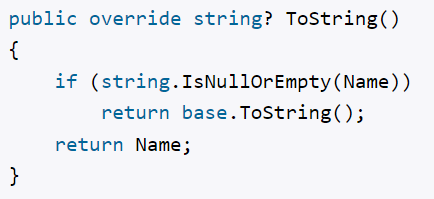
Все классы в .NET, даже те, которые мы сами создаем, а также базовые типы, такие как System.Int32, являются неявно производными от класса Object. Даже если мы не указываем класс Object в качестве базового, по умолчанию неявно класс Object все равно стоит на вершине иерархии наследования. Поэтому все типы и классы могут реализовать те методы, которые определены в классе System.Object. Рассмотрим эти методы.

**1.ToString**

Метод ToString служит для получения строкового представления данного объекта. Для базовых типов просто будет выводиться их строковое значение:

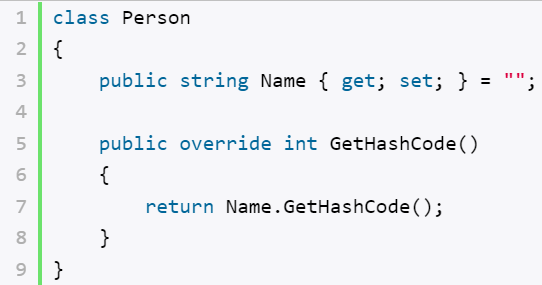


Для переопределения метода ToString(), используется ключевое слово override (как и при обычном переопределении виртуальных или абстрактных методов).



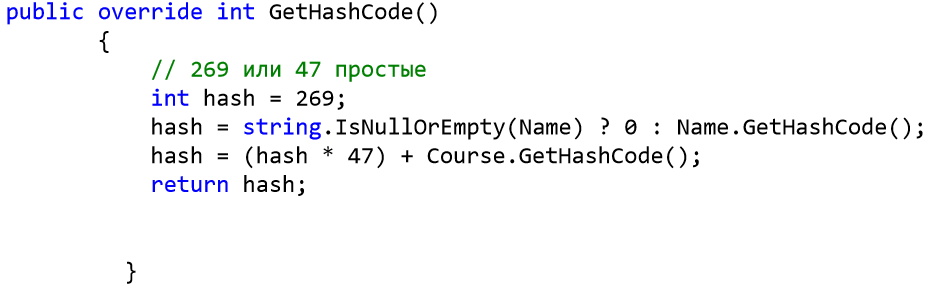
**2.** **GetHashCode**

Метод GetHashCode позволяет возвратить некоторое числовое значение, которое будет соответствовать данному объекту или его хэш-код. По данному числу, например, можно сравнивать объекты. Можно определять самые разные алгоритмы генерации подобного числа или взять реализаци базового типа:



В данном случае метод GetHashCode возвращает хеш-код для значения свойства Name. То есть два объекта Person, которые имеют одно и то же имя, будут возвращать один и тот же хеш-код. Однако в реальности алгоритм может быть самым различным.

**Переопределение:**



**Требования** к GetHashCode

►Случайное распределение

►Не использовать GetHashCode для Object или ValueType (низкая производительность алгоритмов хеширования)

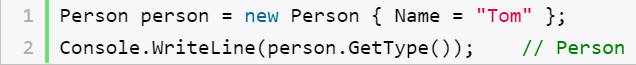
►Использовать экземплярные поля

►Максимально быстрый

►Объекты с одинаковым значением должны возвращать одинаковые коды

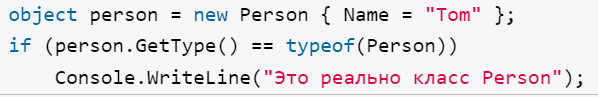
**3.** **GetType**

Метод GetType позволяет получить тип данного объекта:



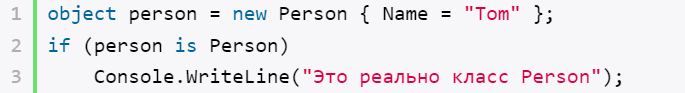
Этот метод возвращает объект Type, то есть тип объекта.

С помощью ключевого слова typeof мы получаем тип класса и сравниваем его с типом объекта. И если этот объект представляет тип Person, то выполняем определенные действия.



Причем поскольку класс Object является базовым типом для всех классов, то мы можем переменной типа object присвоить объект любого типа. Однако для этой переменной метод GetType все равно вернет тот тип, на объект которого ссылается переменная. То есть в данном случае объект типа Person.

Стоит отметить, что проверку типа можно скоратить с помощью оператора is:



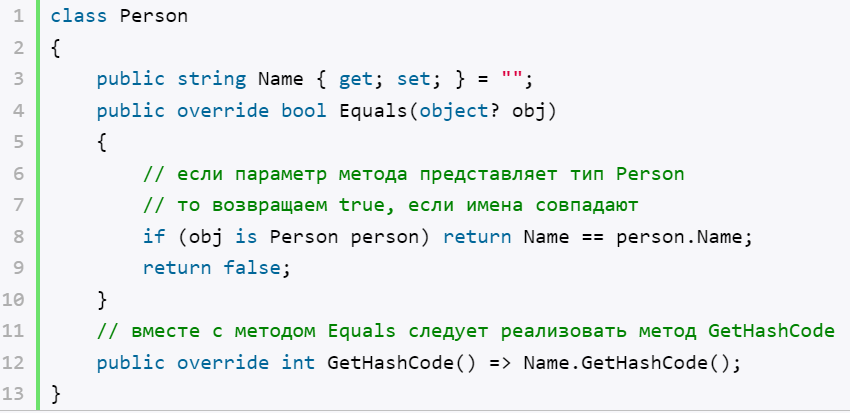
В отличие от методов ToString, Equals, GetHashCode метод GetType() не переопределяется.

**3.** **Equals**

Метод Equals позволяет сравнить два объекта на равенство. В качестве параметра он принимает объект для сравнения в виде типа object и возврашает true, если оба объекта равны:



Рассмотрим пример:



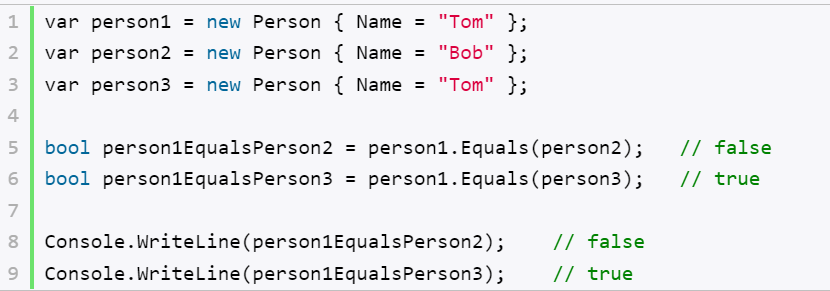
Метод Equals принимает в качестве параметра объект любого типа, который мы затем приводим к текущему классу - классу Person.

Если переданный объект представляет тип Person, то возвращаем результат сравнения имен двух объектов Person. Если же объект представляет другой тип, то возвращается false.

В данном случае для примера применяется довольно простой алгоритм сравнения, однако при необходимости реализацию метода можно сделать более сложной, например, сравнивать по нескольким свойствам при их наличии.

Стоит отметить, что вместе с методом Equals следует реализовать метод GetHashCode.

Применение метода:

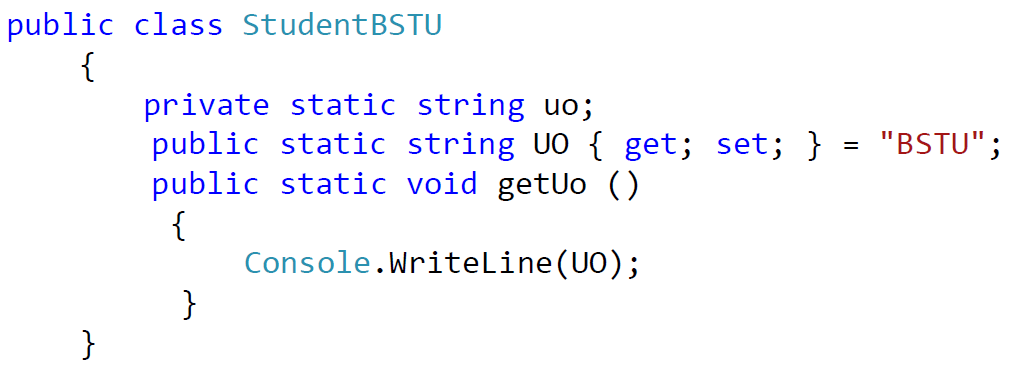


И если следует сравнивать два сложных объекта, как в данном случае, то лучше использовать метод Equals, а не стандартную операцию ==.

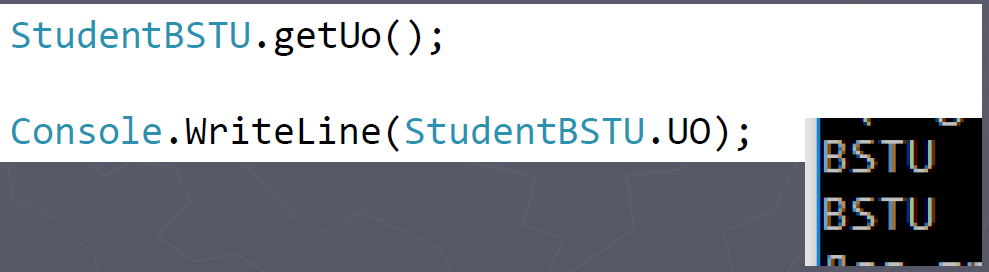
+**Clone() –** есть в лекции Пацей. Создает копию объекта.

**15. Статические методы и статические конструкторы класса.**

-переменные и свойства, которые хранят состояние, общее для всех объектов класса, следует определять как статические   
-методы, которые определяют общее для всех объектов поведение, также следует объявлять как статические



При использовании статических членов необязательно создавать экземпляр класса



Для статических полей будет создаваться участок в памяти, который будет общим для всех объектов класса.

**Свойства статических методов:**

►отсутствует ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта

►в методе static допускается непосредственный вызов только других методов типа static

►для метода static непосредственно доступными оказываются только другие данные типа static, определенные в его классе

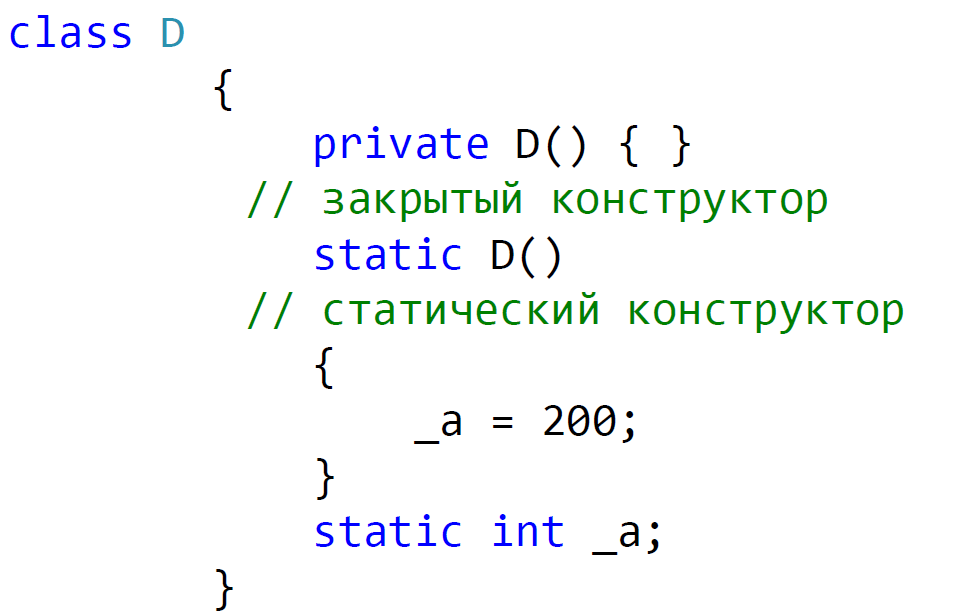
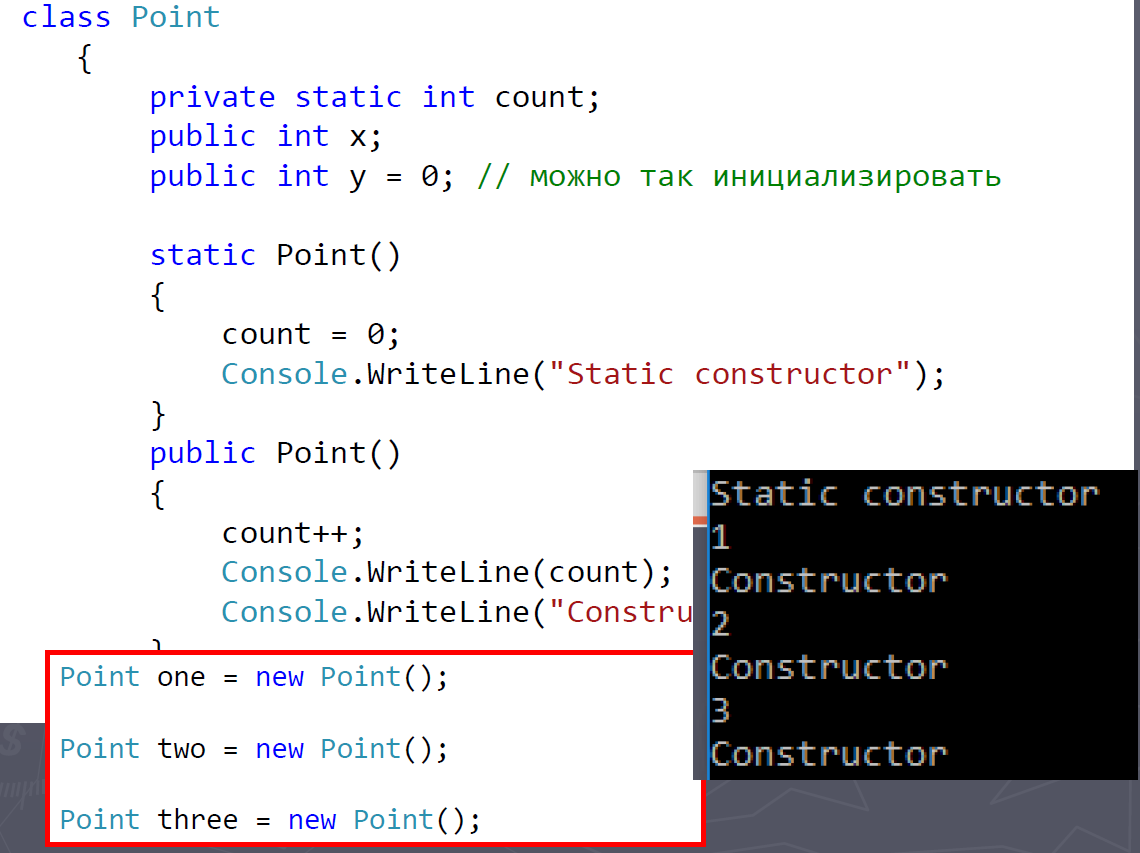
**Статические конструкторы или конструкторы типа.** Конструктор экземпляра инициализирует данные экземпляра  
 конструктор класса (типа)— данные класса.

**Свойства:**

►закрытые автоматически

►не имеет параметров

►нельзя вызвать явным образом (вызываются до создания первого экземпляра объекта или до вызова любого статического метода).

**16. Статические классы. Методы расширения и правила их определения.**

►прямой потомок System.Object

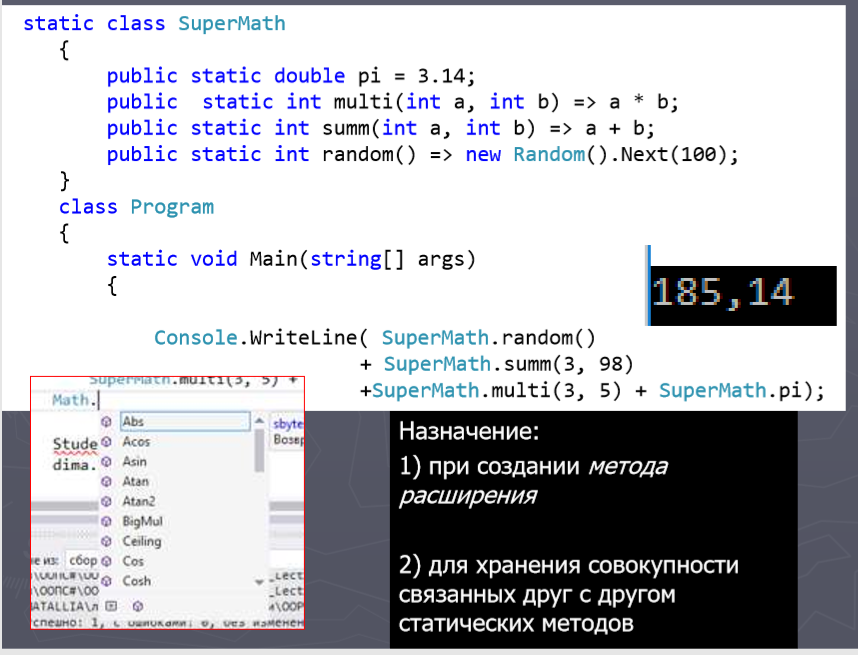
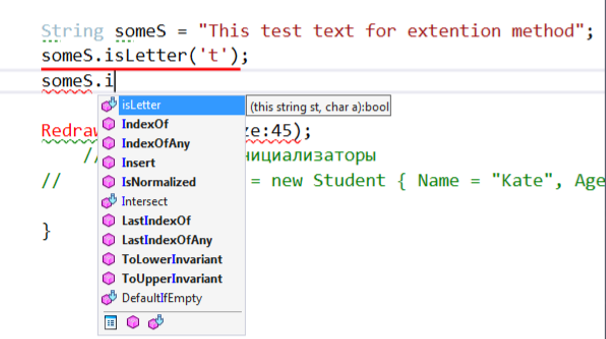
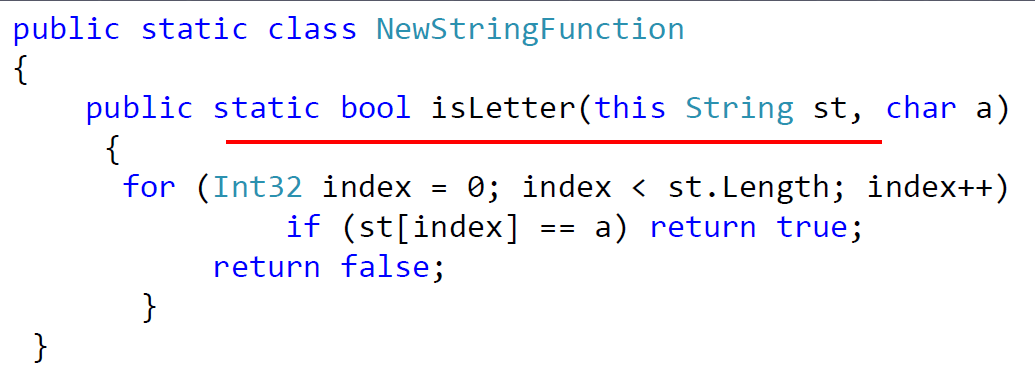
►экземпляры такого класса создавать запрещено

►не должен реализовывать никаких интерфейсов (не вызвать)

►нельзя использовать в качестве поля, параметра метода или локальной переменной

►от него запрещено наследовать

►все элементы такого класса должны явным образом объявляться с модификатором static ►может иметь статический конструктор

►Компилятор не создает автоматически конструктор по умолчанию  
  
**Методы расширения** (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса.  
  
1) Проверяется класс и его базовые

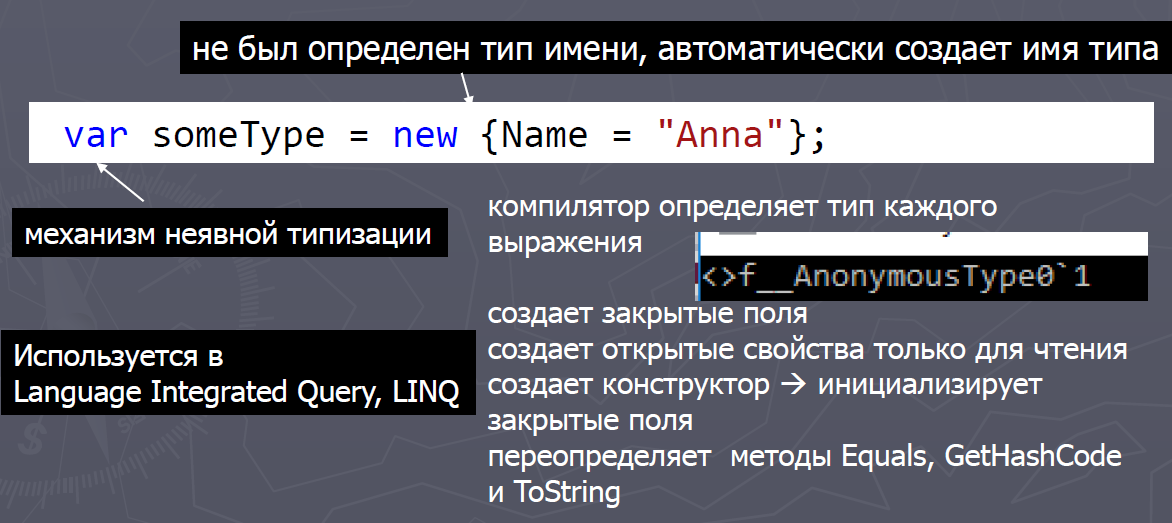
2) Ищется любой статический класс с методом ####, у которого первый параметр соответствует типу выражения ( this)  
  
**Правила для методов расширений**

►1) Методы расширения должны быть объявлены в статическом необобщенном классе (первого уровня)

►2) this перед первым аргументом и только один

►3) использовать аккуратно

**17. Анонимные типы.**

Позволяют создать объект с некоторым набором свойств без определения класса (тип в одном контексте или один раз).  


С# 7  
 Записи - автоматическое создание простых классов

class Person (string name, int age);

**18. Модификаторы параметров - ref , out, params. Необязательные и именованные аргументы.**

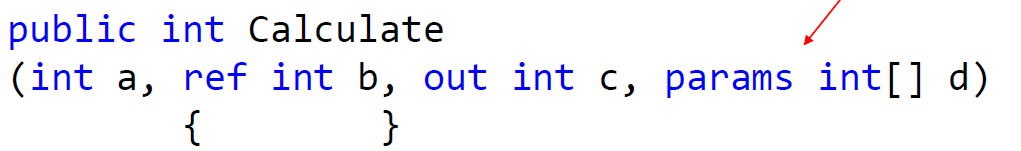
**Для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров:**

►По умолчанию- параметры-значения;

► параметры-ссылки — ref;

► выходные параметры-ссылки — out:

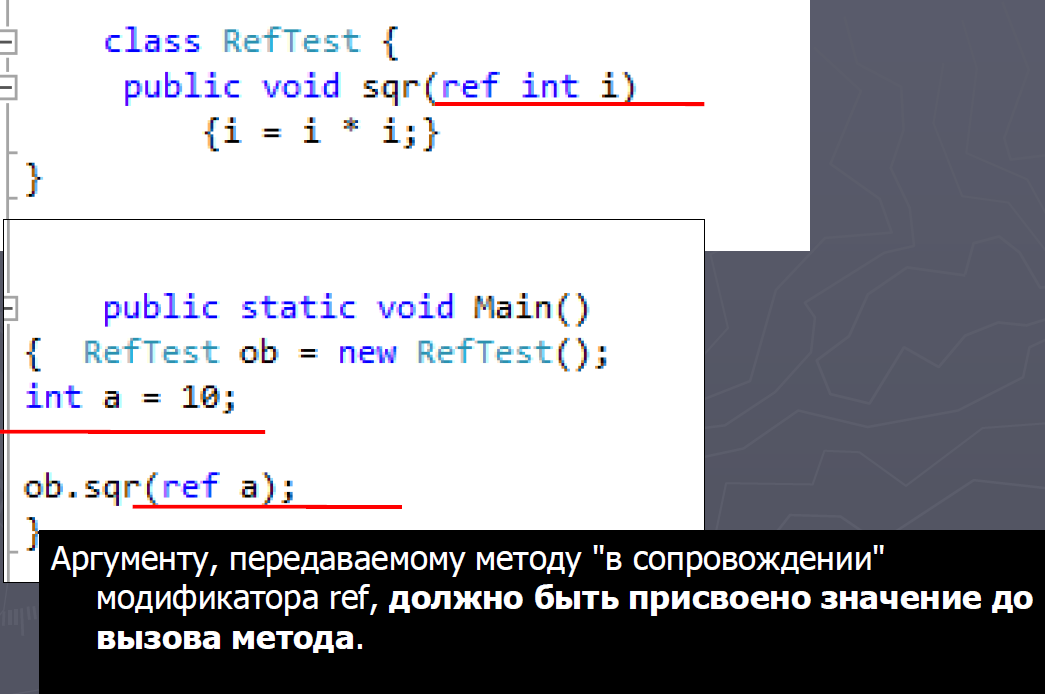
► переменное количество — params (один и последний).

  
**Назначение:**

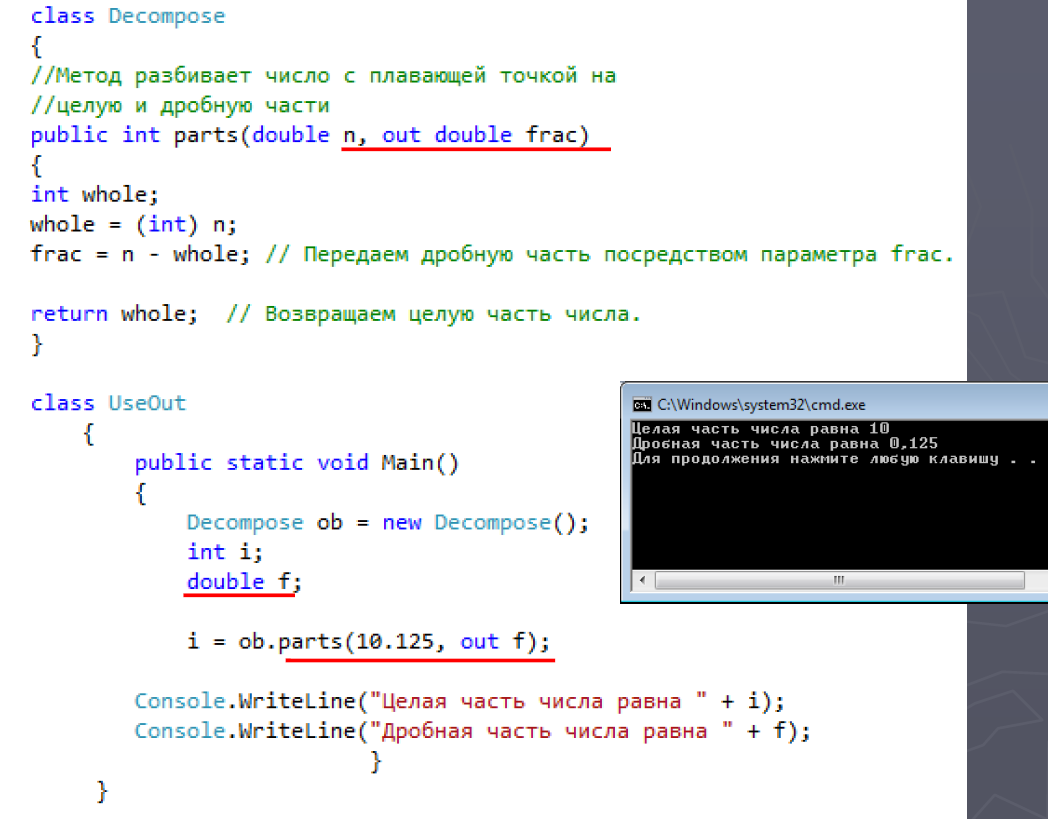
► позволить методу менять содержимое его аргументов

► возвращать более одного значения

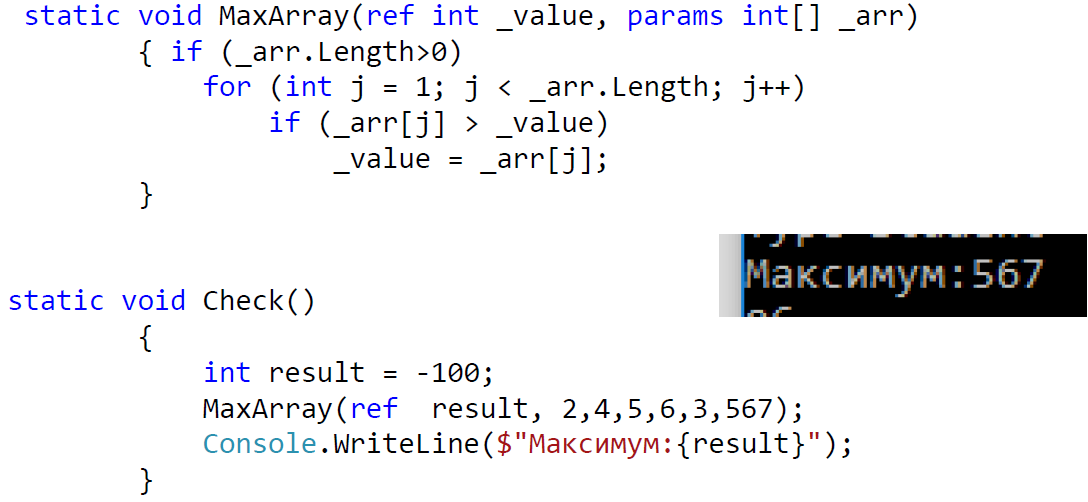
**ref заставляет С# организовать вместо вызова по значению вызов по ссылке**Аргументу, передаваемому методу "в сопровождении" модификатора ref, должно быть присвоено значение до вызова метода.



**Модификатор out** подобен модификатору ref за одним исключением: его можно использовать для передачи значения из методаout-параметр "поступает" в метод без начального значения, но метод (до своего завершения) обязательно должен присвоить этому параметру значение

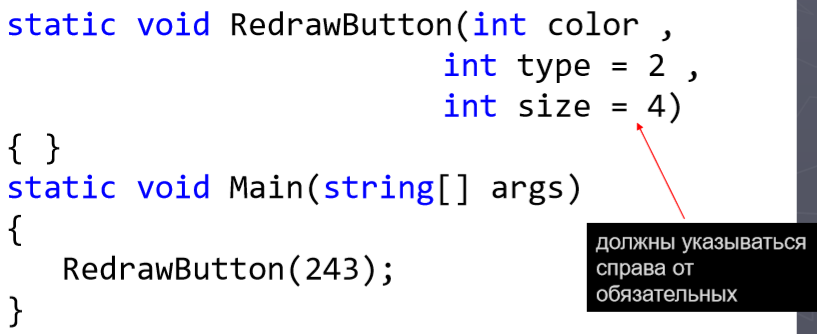


**params** позволяет передавать методу переменное количество аргументов одного типа

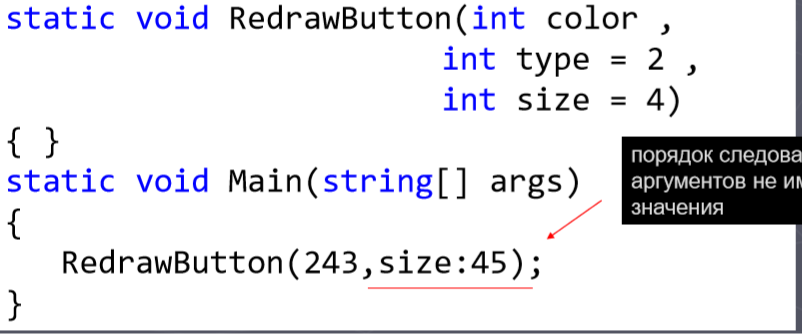


**Необязательные аргументы**

►позволяет определить используемое по умолчанию значение для параметра метода

►можно применять в конструкторах, индексаторах  


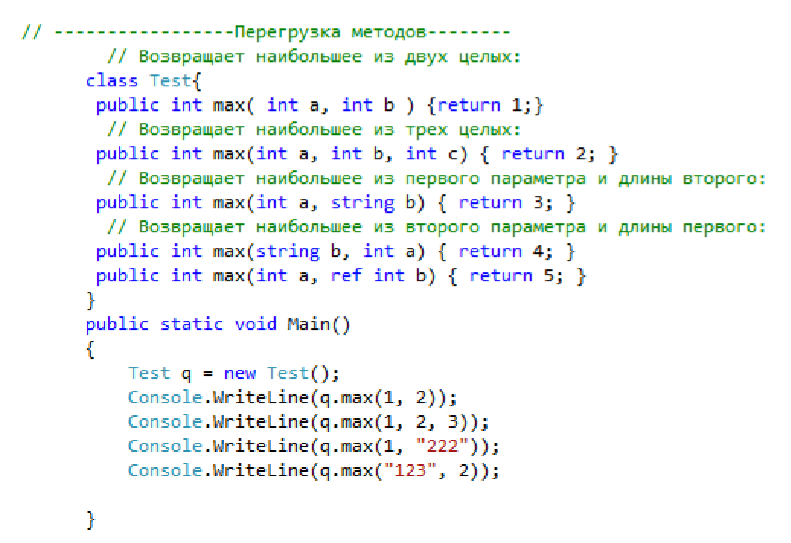
**Именованные аргументы**  
позволяет указать имя того параметра, которому присваивается его значение ( в конструкторах, индексаторах или делегатах.)



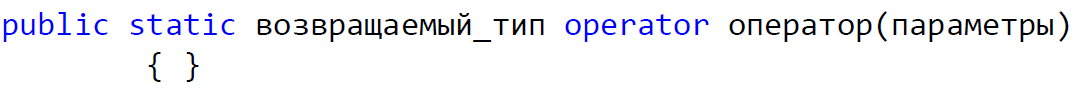
**19. Перегрузка методов и операторов. Правила перегрузки операторов.**

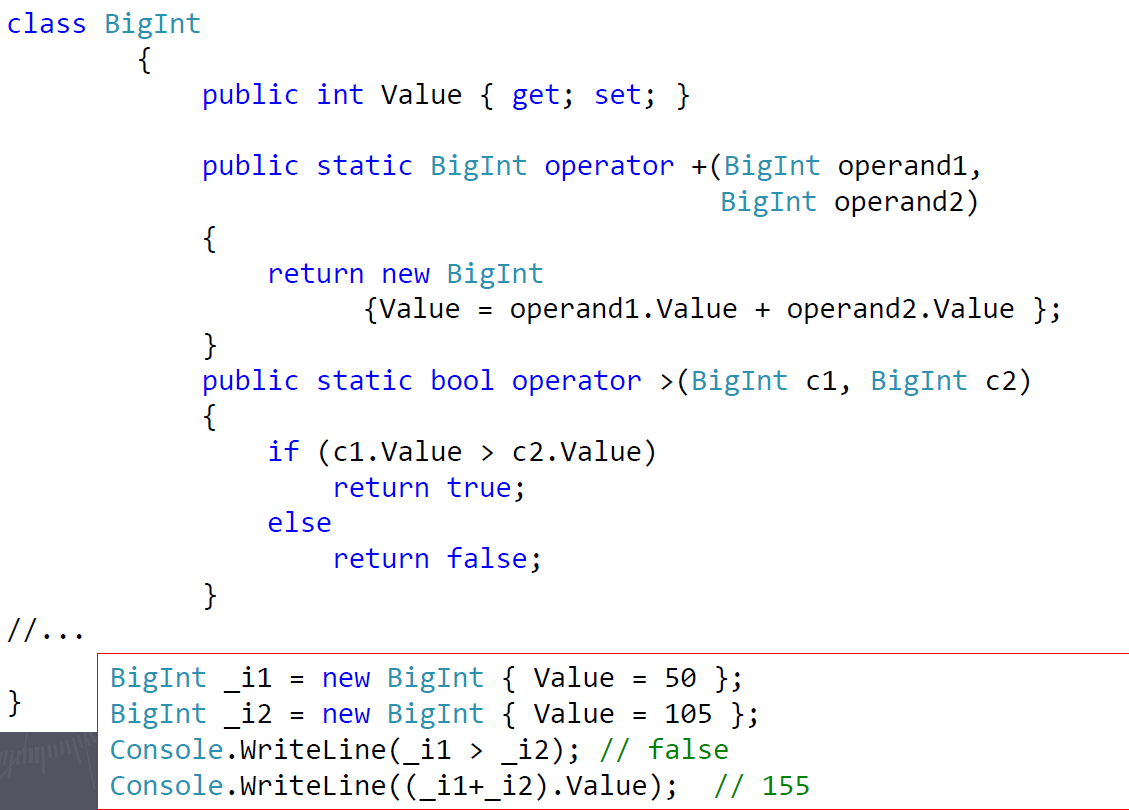
**Перегрузка методов**

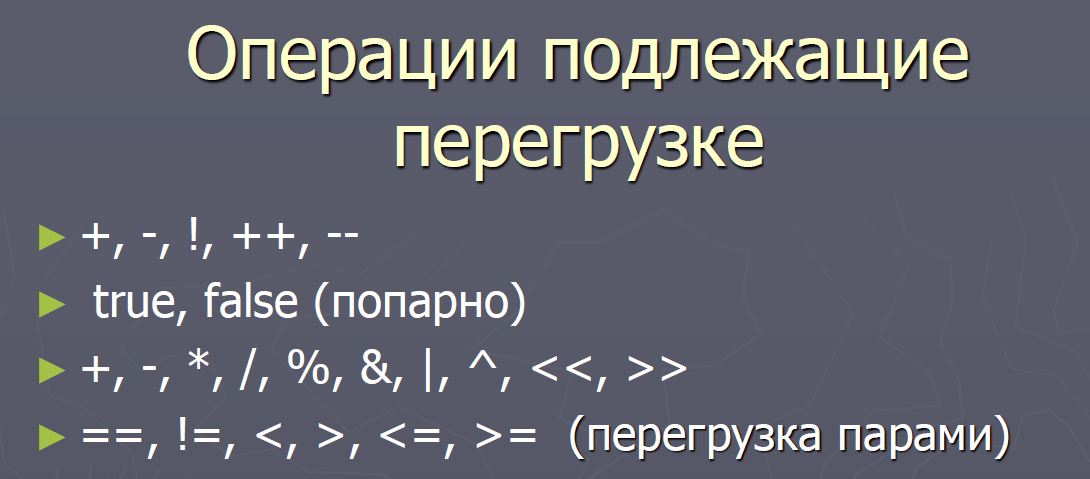
**►**один и тот же метод, но с разным набором параметров ► позволяет обращаться к связанным методам посредством одного, общего для всех имени.

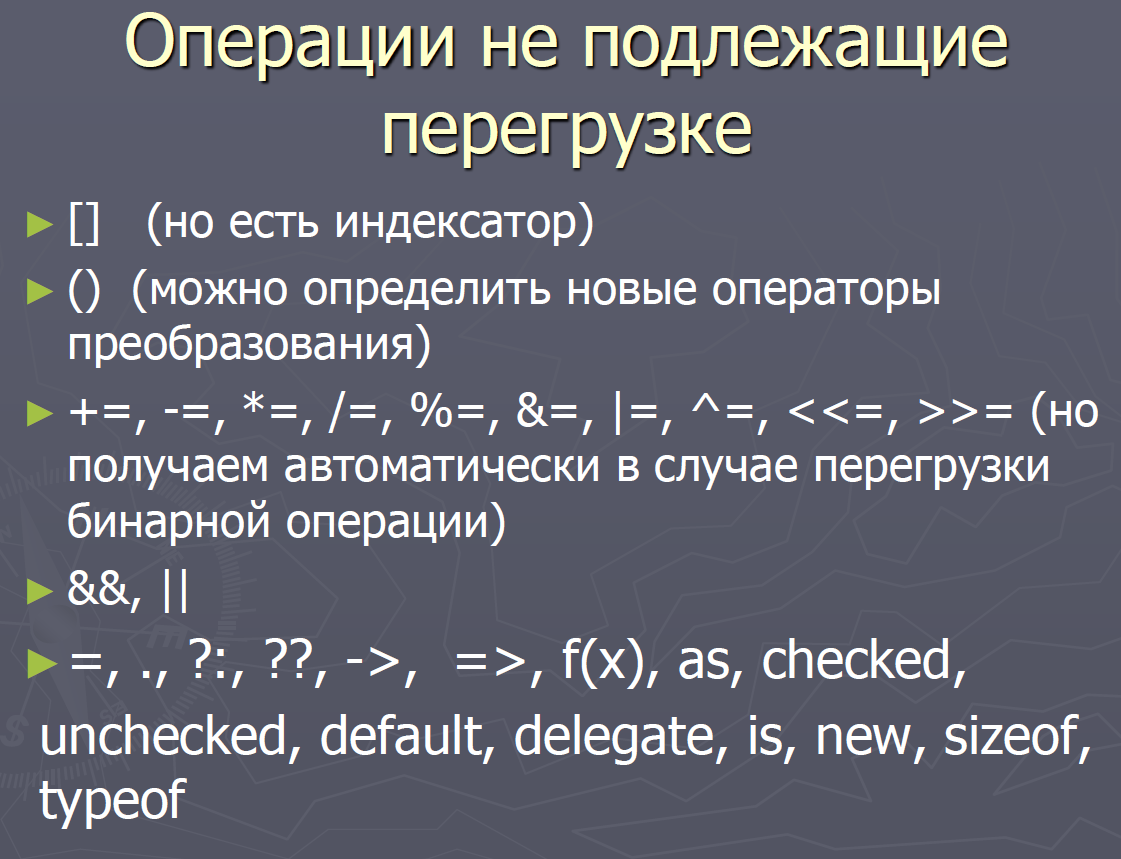
►никакие два метода внутри одного и того же класса не должны иметь одинаковую сигнатуру сигнатура (signature) = имя метода + список его параметров (не включает тип значения, возвращаемого методом, не включает paramsпараметр)  


**Перегрузка операций**►способ объявления новых операций для типа  
**Спецификация CLR требует, чтобы перегруженные операторные методы были**1) открытыми и статическими  
2) тип одного из параметров или возвращаемого значения совпадал с типом, в котором определен операторный метод









**Правила:**

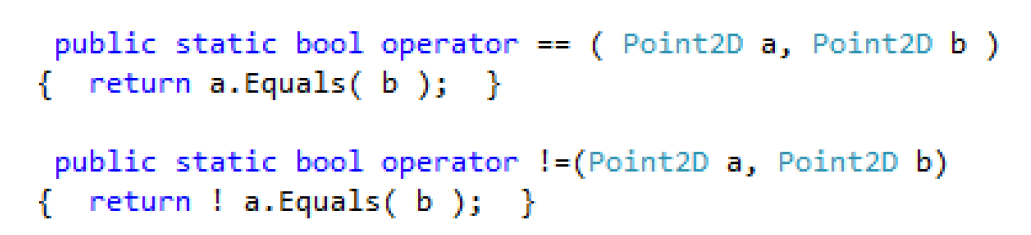
►префиксные операции ++ и – – перегружаются парами;

►операции сравнения перегружаются парами: == и != ; < и >;<= и >=.

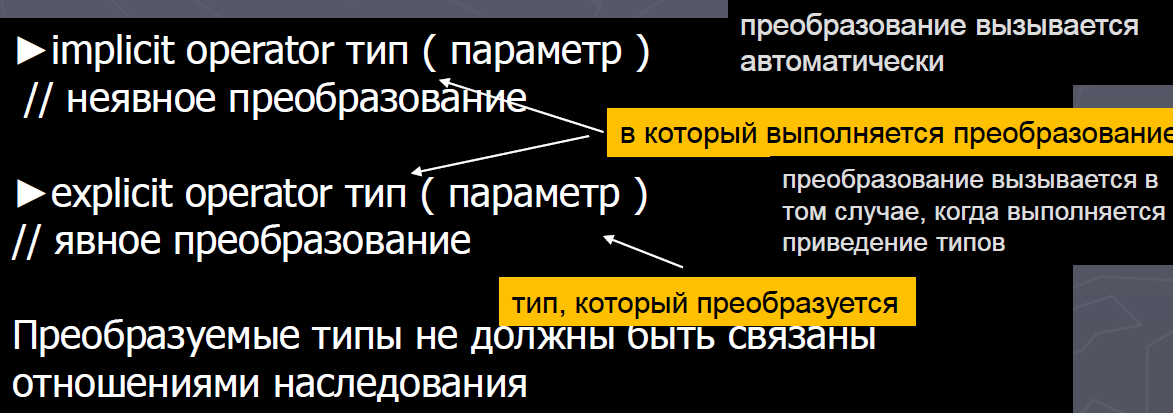
►Перегруженные операции обязаны возвращать значения

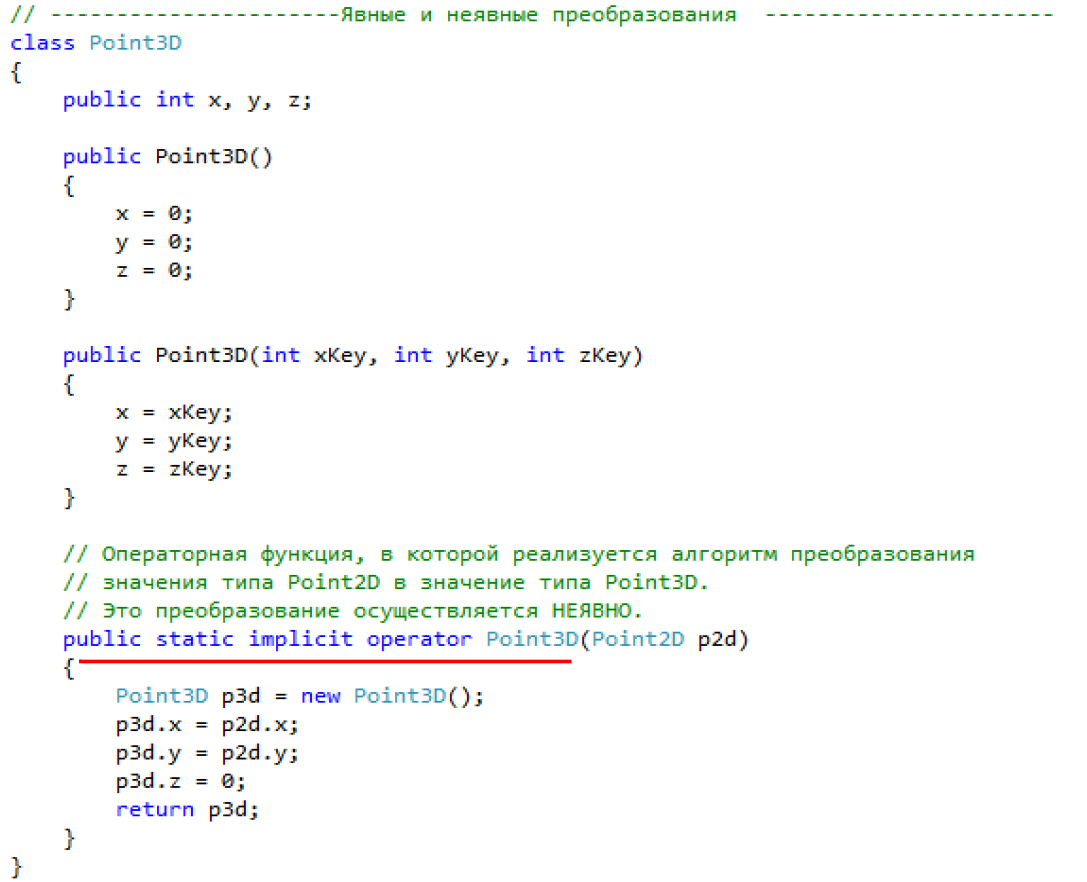
►Должны объявляться как public и static

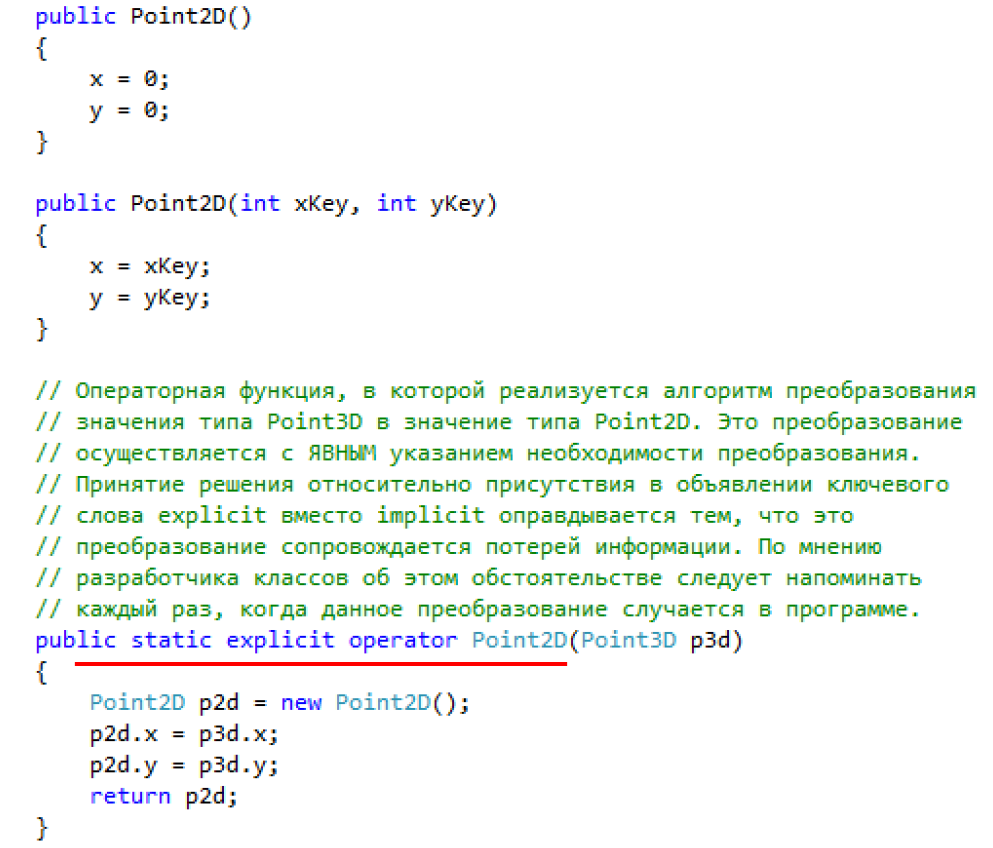
►префиксная и постфиксная формы операций ++ и --, в отличие от оригинальных операций, семантически НЕ различаются.

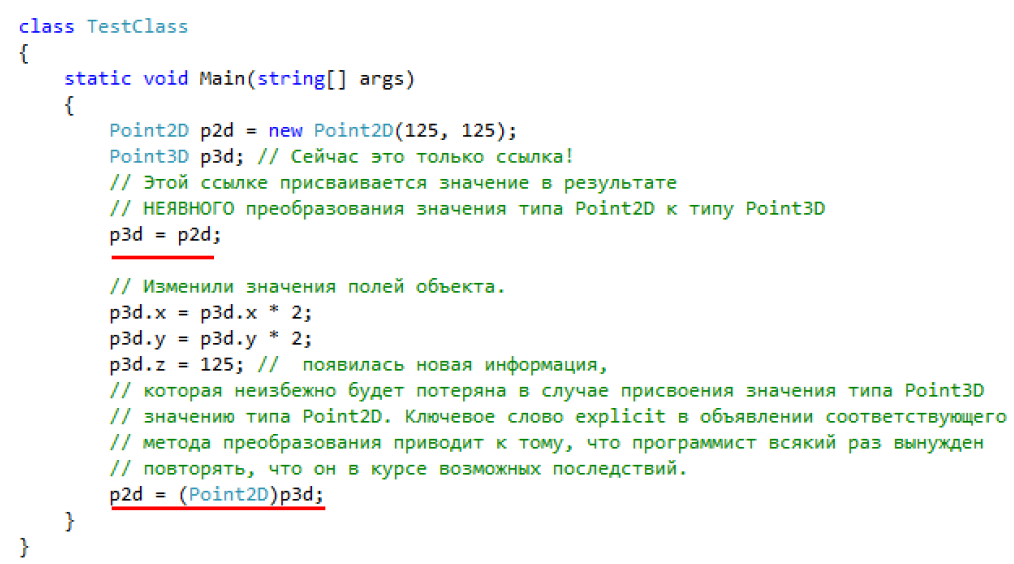


**20. Операции преобразования типа. Явная и неявная форма. Ограничения.**

**Операции преобразования типа**►преобразует объект исходного класса в другой тип   
►явная и неявна форма - будет ли этот алгоритм выполняться неявно или необходимо 







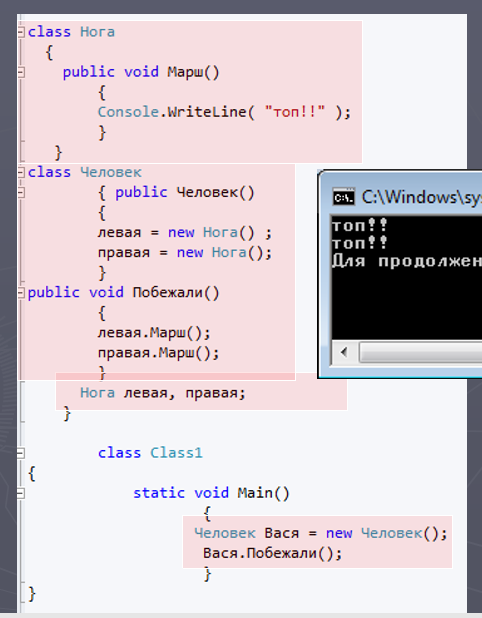
►Ключевые слова implicit и explicit в сигнатуру не включаются  
  
**Ограничения на операторы преобразования**   
►Исходный или целевой тип преобразования должен относиться к классу, для которого объявлено данное преобразование   
►Нельзя указывать преобразование в/из класс object или же из этого класса   
►Для одних типов данных нельзя указывать одновременно явное и неявное преобразование   
►Нельзя указывать преобразование базового класса в производный класс   
►Нельзя указывать преобразование в/из интерфейс

**21. Вложенные типы. Вложенные объекты**

Тип, определенный внутри типа называется вложенным типом  
**class Person {  
 public class Date {  
 public Date() { }   
 }   
}   
Person.Date today = new Person.Date();**

По умолчанию являются private  
Вложенный тип может получить доступ к внешнему типу, а внутренний тип — к внешнему   
Вложенный тип имеет доступ ко всем членам, которые доступны вмещающему типу

**Вложенные объекты**

***public class Date {   
public Date() { }   
}   
class Person {   
Date birthday;  
}  
Person anna = new Person(); anna.birthday = null;*  
  
Вложение или включение классов модель включения-делегирования.**

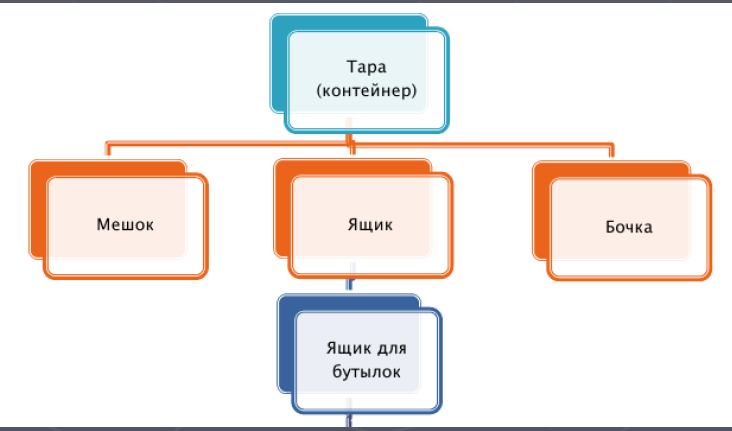
**#region … -** Сворачивание и разворачивания блоков кода

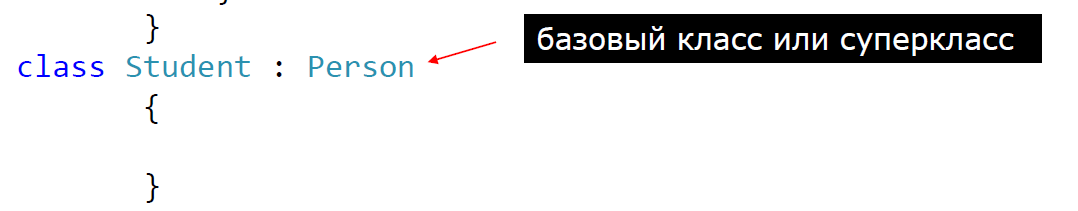
**#endregion …**

**22. Правила наследования C#.  
Наследование -** это механизм получения нового класса на основе уже существующего  
**Роль наследования**

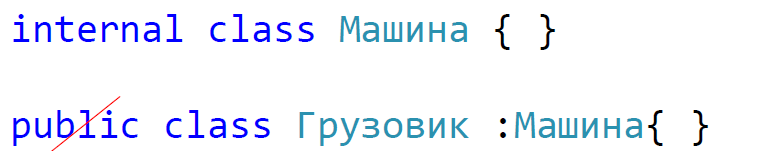
**►**формирует иерархию

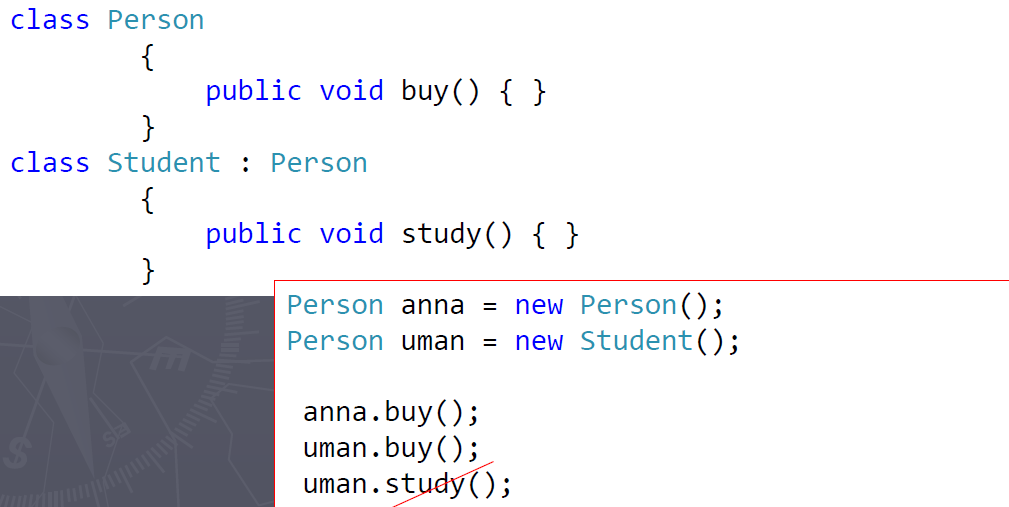
►поощряет повторное использование кода

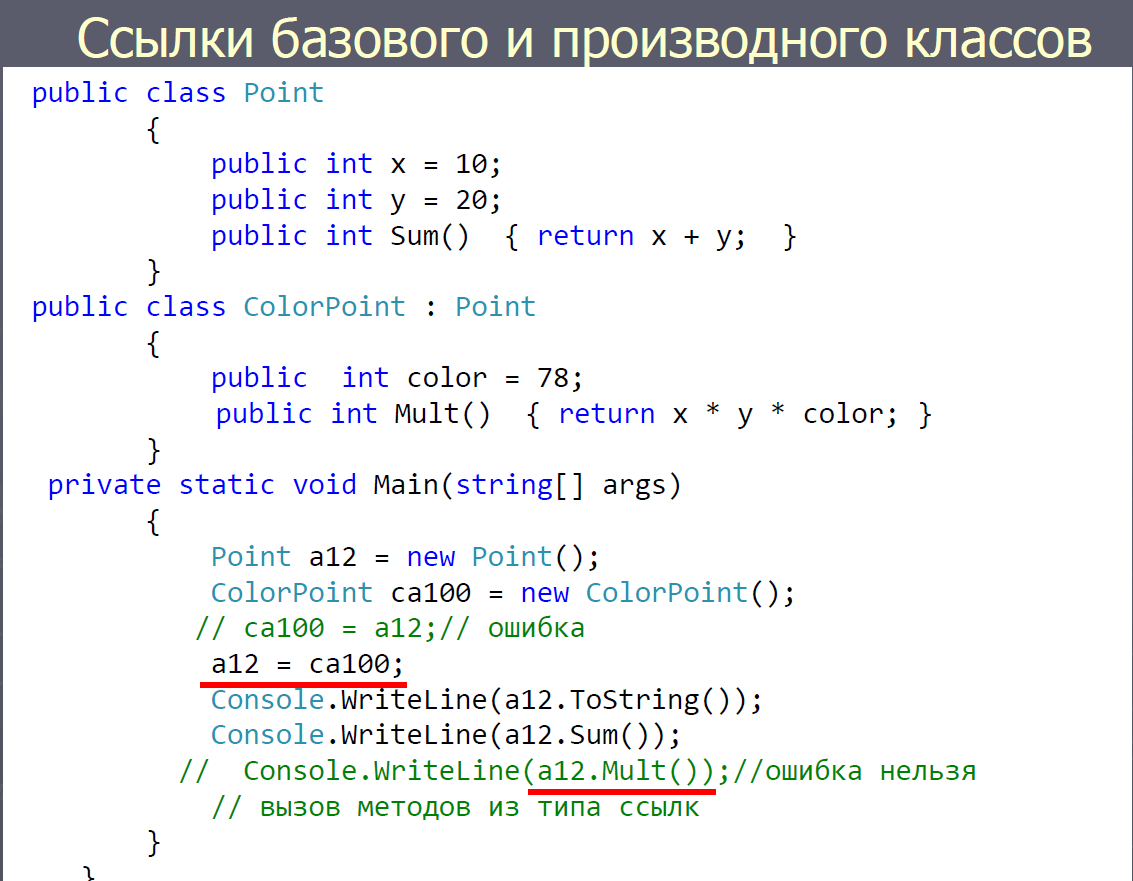


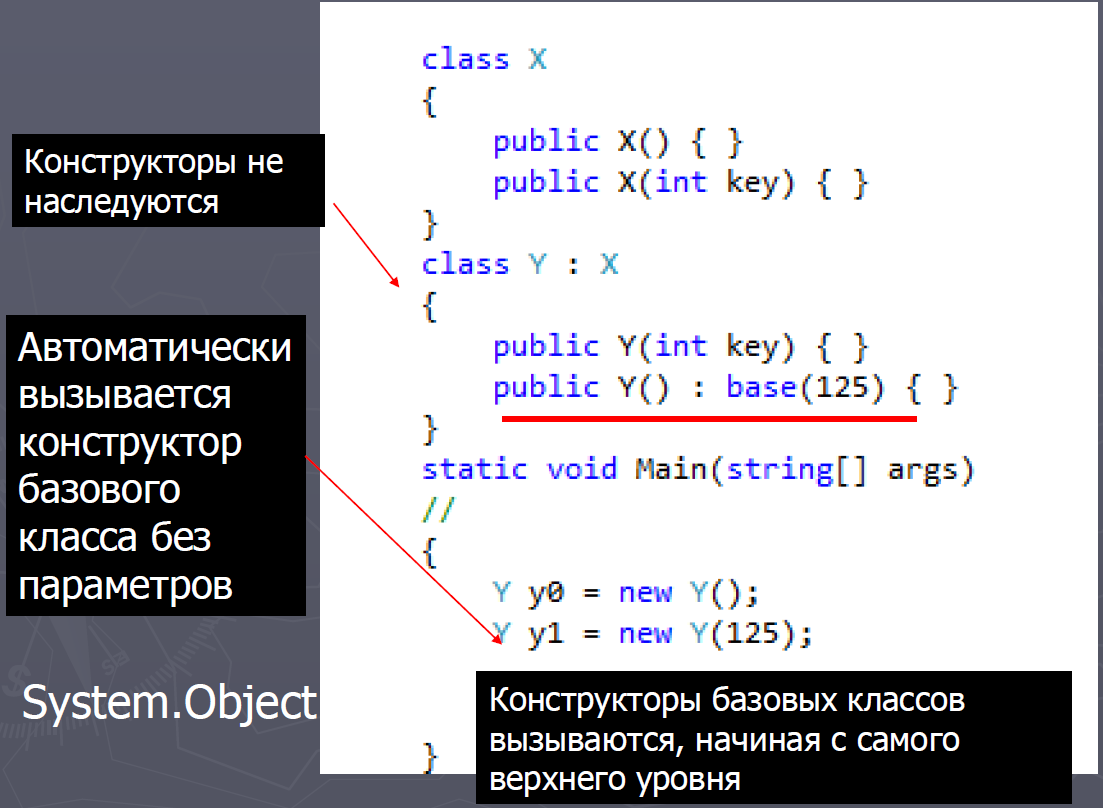


**Правила наследования:**1) В C# наследование всегда подразумевается открытым  
2) Запрещено множественное наследование классов (но не интерфейсов)   
3) наследуются все свойства, методы, поля и т.д., которые есть в базовом классе   
4) Производному классу доступны public, internal, protected и protected internal члены базового класса (private – недоступны)  
5) не наследуются конструкторы базового класса   
6) тип доступа к производному классу должен быть таким же, как и у базового класса или более строгим

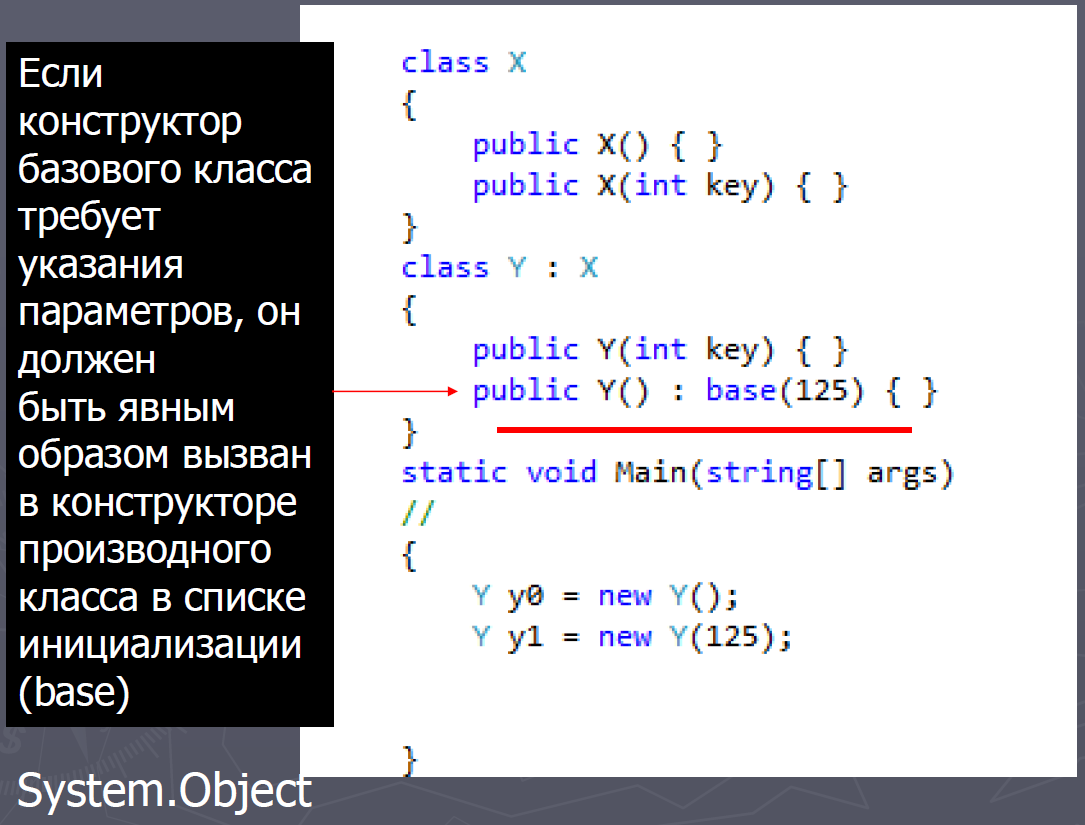
  
7) Ссылке на объект базового класса можно присвоить объект производного класса (но вызываются для него только методы и свойства, определенные в базовом классе.)





**Ключевое слово base**





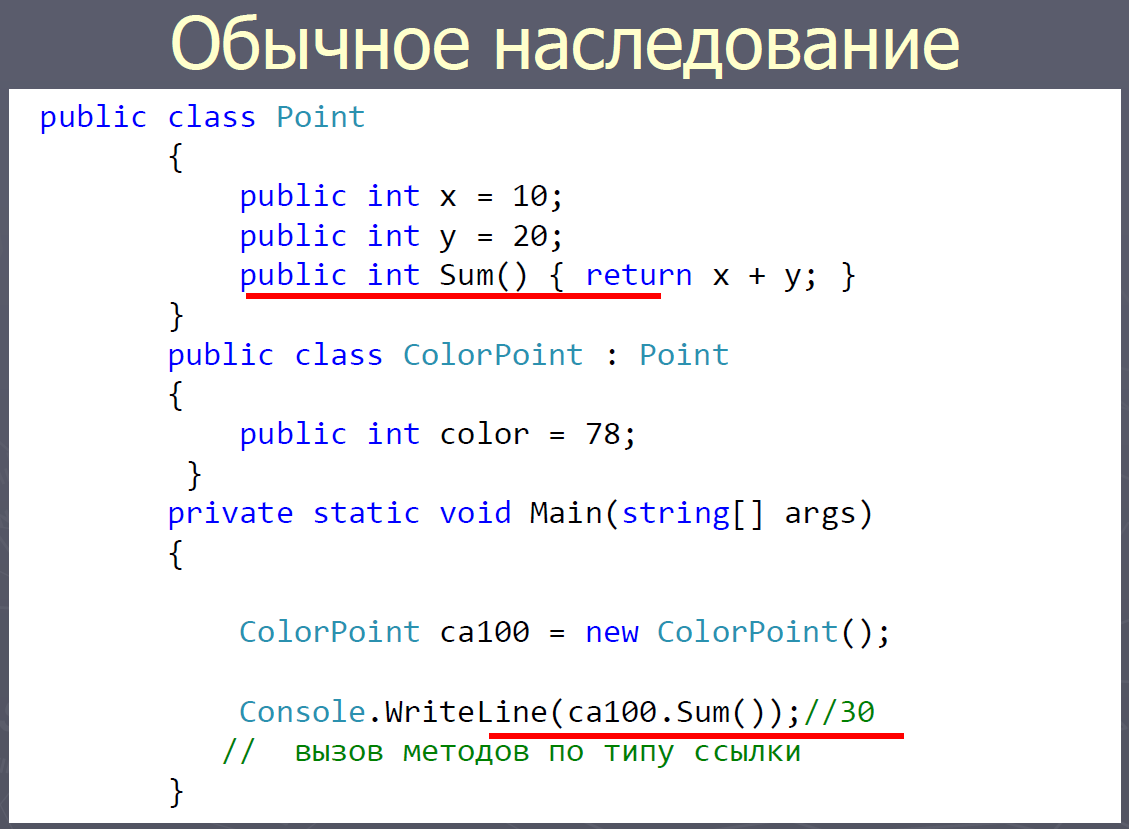
**23. Сокрытие имен при наследовании. Обращение к срытым членам.Стратегия наследования**

**Стратегии наследования**

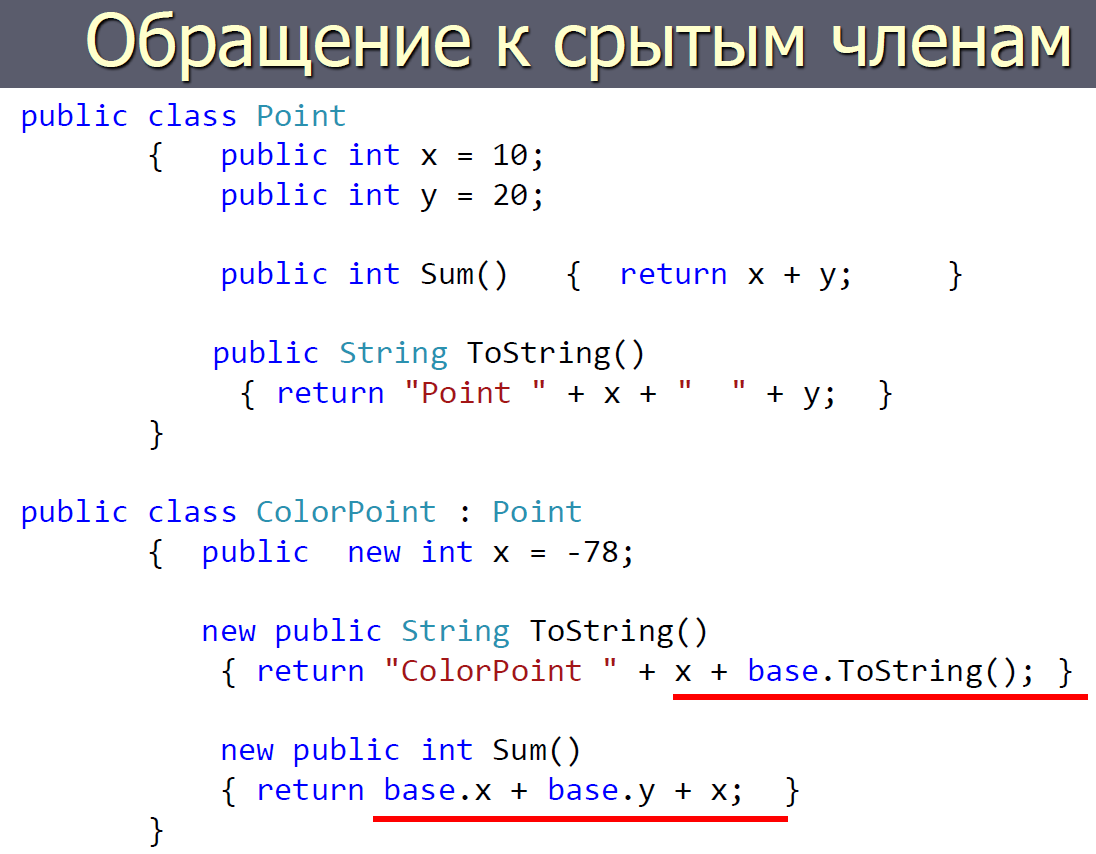
►Обычное наследование всех членов базового класса в классе-наследнике

►Переопределение членов базового класса в классе-наследнике (полиморфизм)

►Сокрытие членов базового класса в классе-наследнике

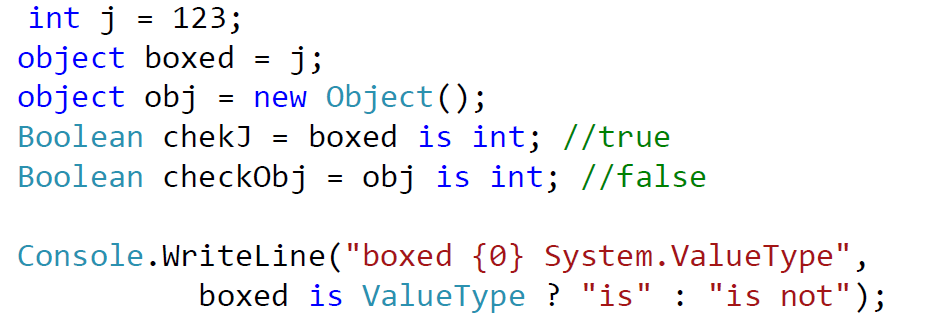






**24. Использование операций is и as**

**Операции is**Возвращает булевское значение, говорящее о том, можете ли вы преобразовать данное выражение в указанный тип  
**Оператор is никогда не генерирует исключение.**



**Операция as**позволяет преобразовывать тип в определенный ссылочный тип с применением следующего синтаксиса:  ***операнд as <тип>***

**Выполняется**►Если <операнд> имеет тип, заданный в <тип>.

► Если <операнд>, может быть неявно преобразован в <тип>.

►Если операнд <операнд>, может быть упакован в <тип>.  
