1. Назовите принципы ООП. Поясните каждый из них.

*Ответ:* Инкапсуляция, наследование, полиморфизм и абстракция данных – принципы ООП.

**Инкапсуляция** – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали  
реализации от пользователя. Механизм связывает вместе данные и  
код, обрабатывающий эти данные, и сохраняет их от внешнего  
воздействия и ошибочного использования. Никто не знает, что внутри, и никто не может менять данные снаружи.

Свойства инкапсуляции: совместное хранение данных и функций, сокрытие внутренней информации от пользователя, изоляция пользователя от особенностей реализации.

**Полиморфизм** –  возможность объектов с одинаковой спецификацией иметь различную реализацию. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций». Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс.

Это – способность вызывать метод потомка через экземпляр предка.

Поддержка полиморфизма осуществляется через виртуальные функции, механизм перегрузки функций и операторов, а также обобщения.

**Наследование** – механизм, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс).

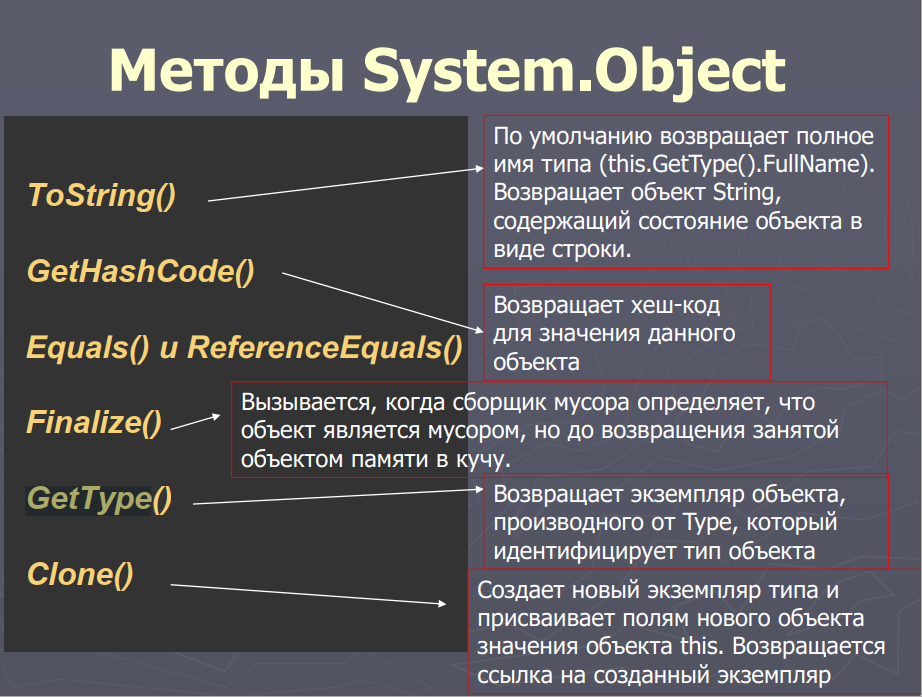
**Абстракция** – механизм, который подразумевает разделение и независимое рассмотрение интерфейса и реализации. Это использование только тех характеристик [объекта](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которые с достаточной точностью представляют его в данной системе. Основная идея состоит в том, чтобы представить объект минимальным набором полей и методов и при этом с достаточной точностью для решаемой задачи.

2. Назовите класс .NET, от которого наследуются все классы.

*Ответ:* **System.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.

Класс System.Object предоставляет низкоуровневые службы для производных классов. Является исходным базовым классом для всех классов .NET и корнем иерархии типов.

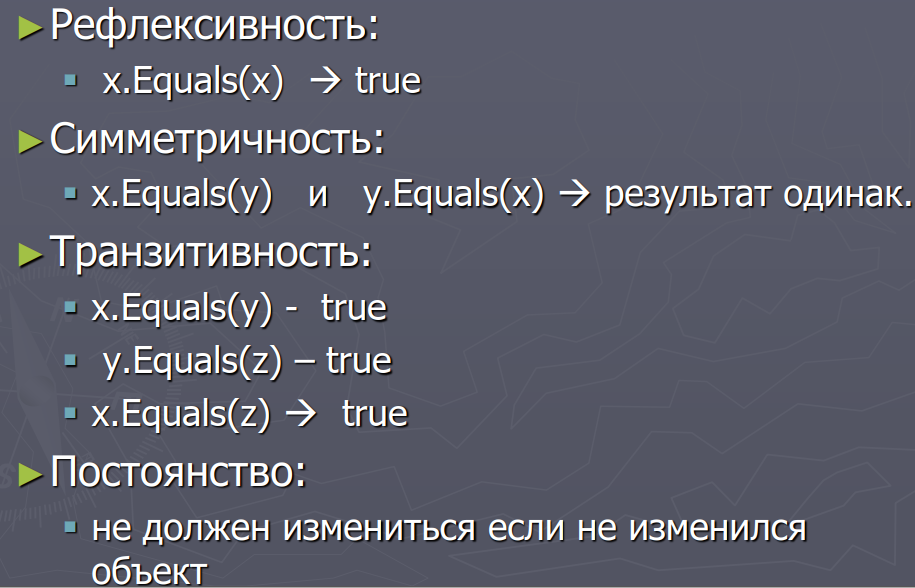
Класс System.Object гарантирует, что каждый экземпляр типа обладает минимальным набором аспектов поведения (методы ToString(), Equals(), ReferenceEquals(), GetHashCode(), GetType(), Finalize(), Clone()).



3. Охарактеризуйте открытые методы System.Object.

*Ответ:* Открытые методы возможно переопределять в объектах-потомках.

**Equals()** – Возвращает true, если два объекта имеют одинаковые значения. Метод Equals принимает в качестве параметра объект любого типа, который мы затем приводим к текущему, если они являются объектами одного класса. Затем сравниваем по именам. Если имена равны, возвращаем true, что будет говорить, что объекты равны.

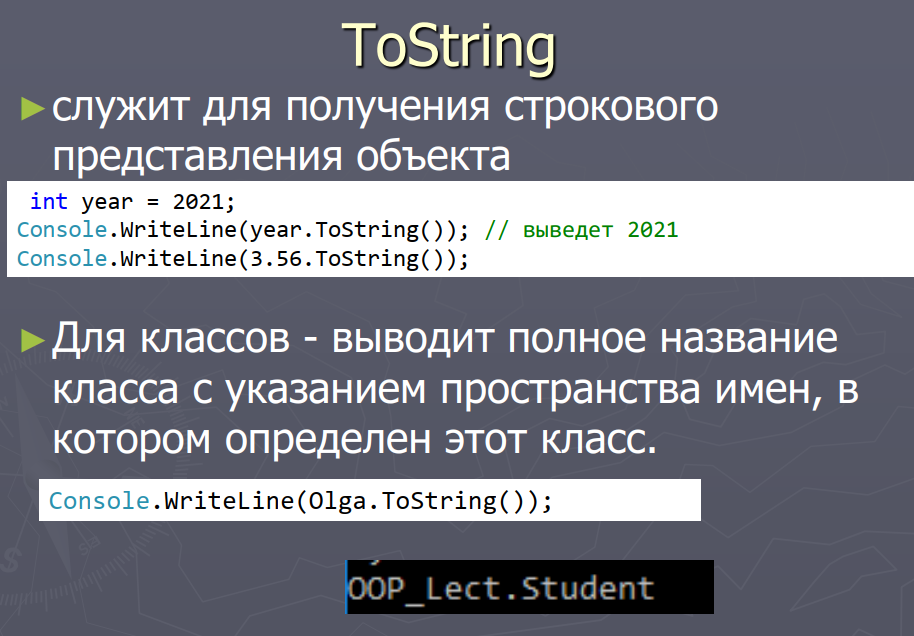


**ReferenceEquals ()** – Для проверки на тождественность нужно всегда вызывать ReferenceEquals (то есть проверять на предмет того, относятся ли две ссылки к одному объекту).

Разработчики FCL решили, что было бы чрезвычайно полезно иметь возможность добавления в хеш-таблицы любых экземпляров любых типов. С этой целью в System.Object включен виртуальный метод **GetHashCode()**, позволяющий вычислить для любого объекта целочисленный (Int32) хеш-код.

**GetHashCode ()** – Возвращает хэш-код для данного объекта. Хэш-код предназначен для эффективной вставки и уточняющего запроса в коллекциях, основанных на хэш-таблице. Этот метод используется, когда объект помещается в структуру данных, известную как карта (map), которая также называется [хеш-таблицей](https://professorweb.ru/my/csharp/charp_theory/level12/12_5.php) или словарем. Применяется классами, которые манипулируют этими структурами, чтобы определить, куда именно в структуру должен быть помещен объект.

**ToString()** – метод служит для для получения строкового представления объекта. **Для классов – выводит полное название класса с указанием пространства имен, в котором определен этот класс**. Для базовых типов – будет выводиться их строковое значение.



**GetType()** - возвращает экземпляр объекта, производного от Type, который идентифицирует тип объекта, вызвавшего GetType(). Этот метод возвращает экземпляр класса, унаследованный от System.Type. Этот объект может предоставить большой объем информации о классе, членом которого является ваш объект, включая базовый тип, методы, свойства.

4. Охарактеризуйте закрытые методы System.Object.

*Ответ:* **Clone()** – Этот **невиртуальный (нельзя переопределять)** метод создает новый экземпляр типа и присваивает полям нового объекта соответствующие значения объекта this. Возвращается ссылка на созданный экземпляр. Этот метод является **защищенным**, а потому не может вызываться для копирования внешних объектов.

**Finalize()** – Этот **виртуальный** метод вызывается, когда сборщик мусора определяет, что объект является мусором, но до возвращения занятой  
объектом памяти в кучу.

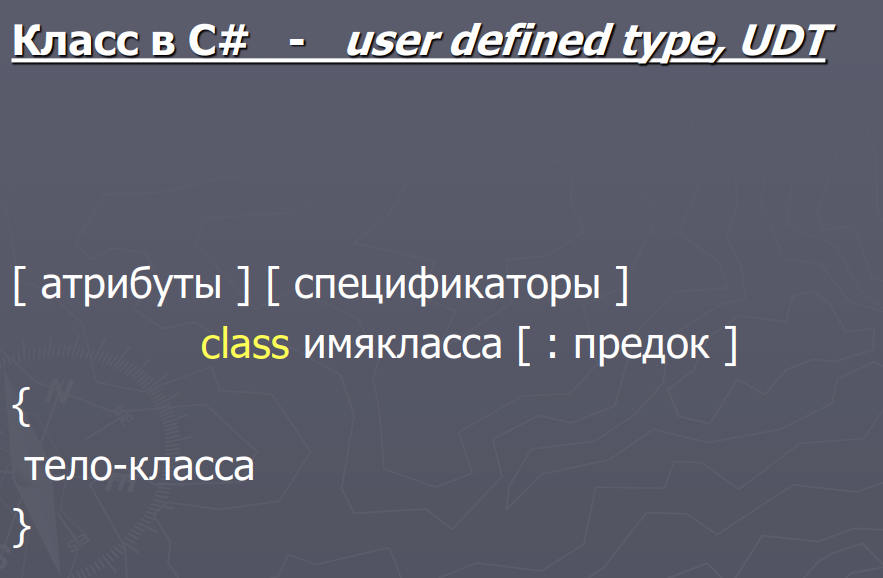
**В типах, требующих очистки при сборке мусора, следует переопределить этот метод**.

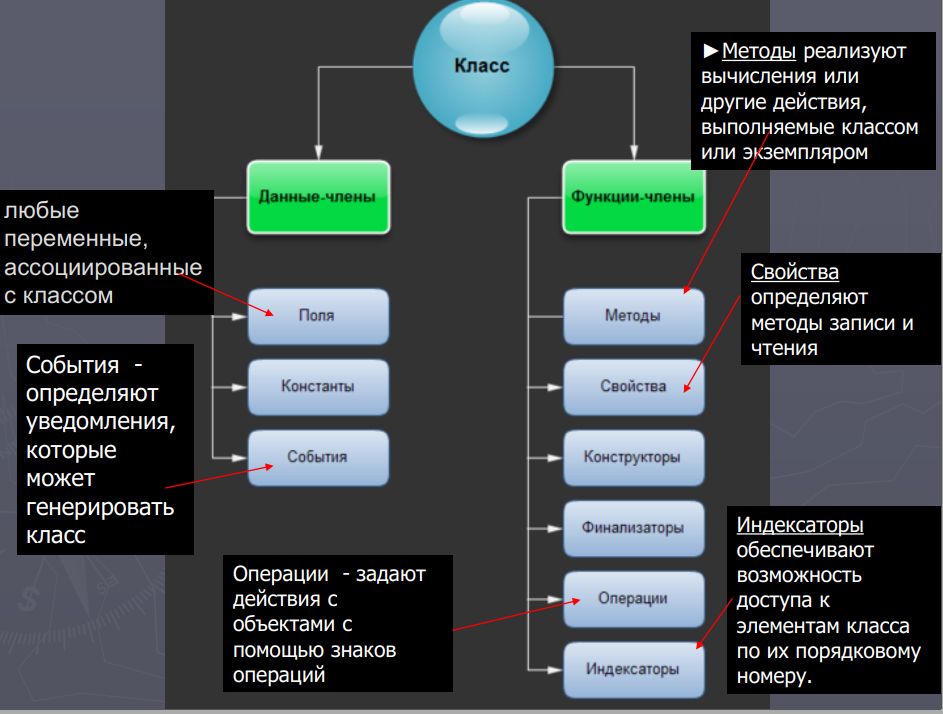
Назначение этого метода в C# примерно соответствует деструкторам С++, и **он вызывается при сборке мусора для очистки ресурсов, занятых ссылочным объектом**. **Реализация Finalize() из Object на самом деле ничего не делает и игнорируется сборщиком мусора**. Обычно переопределять Finalize() необходимо, если объект владеет неуправляемыми ресурсами, которые нужно освободить при его уничтожении. Сборщик мусора не может сделать это напрямую, потому что он знает только об управляемых ресурсах, поэтому полагается на финализацию, определенную вами.

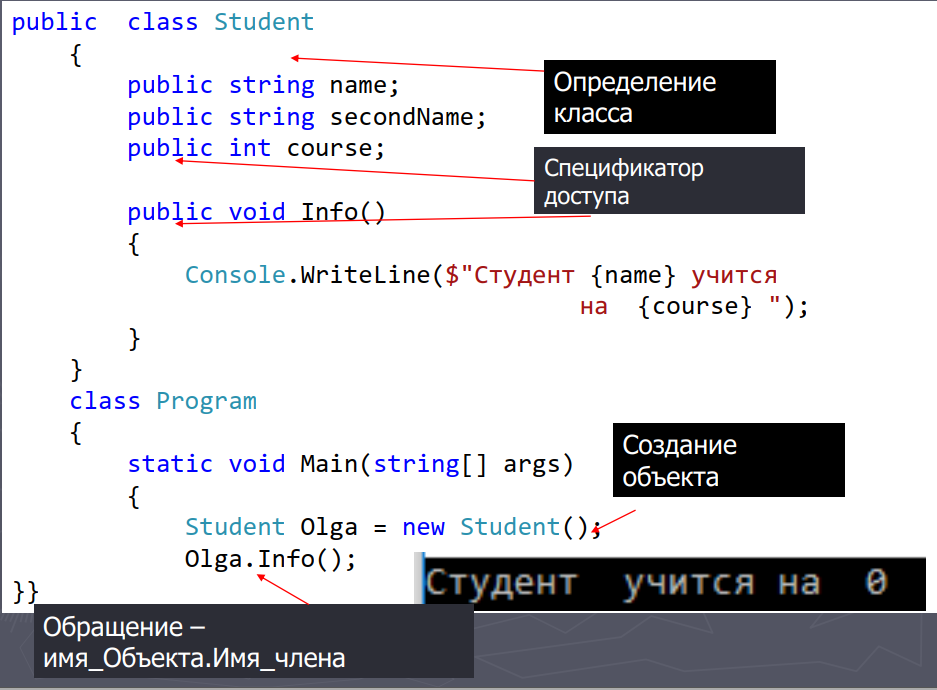
5. Приведите пример определения класса.

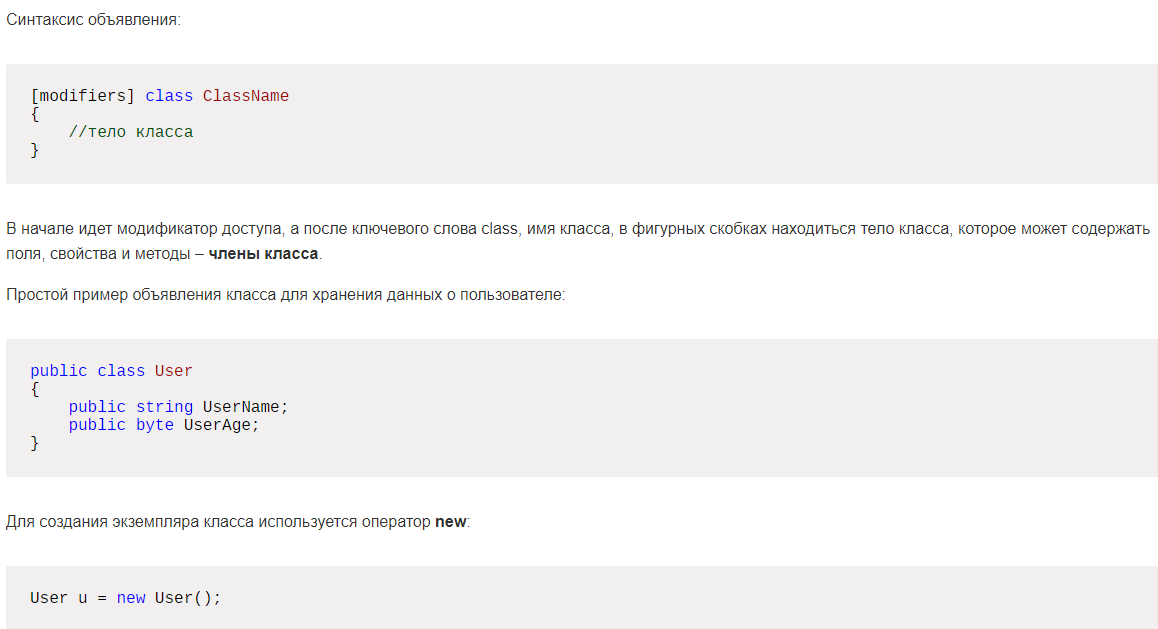
*Ответ:* **Класс** – это абстрактное понятие - шаблон, по которому определяется форма объекта. Класс — это определение общего поведения для объектов, который этот класс порождает.

**Объект** – это физическая реализация класса (шаблона).



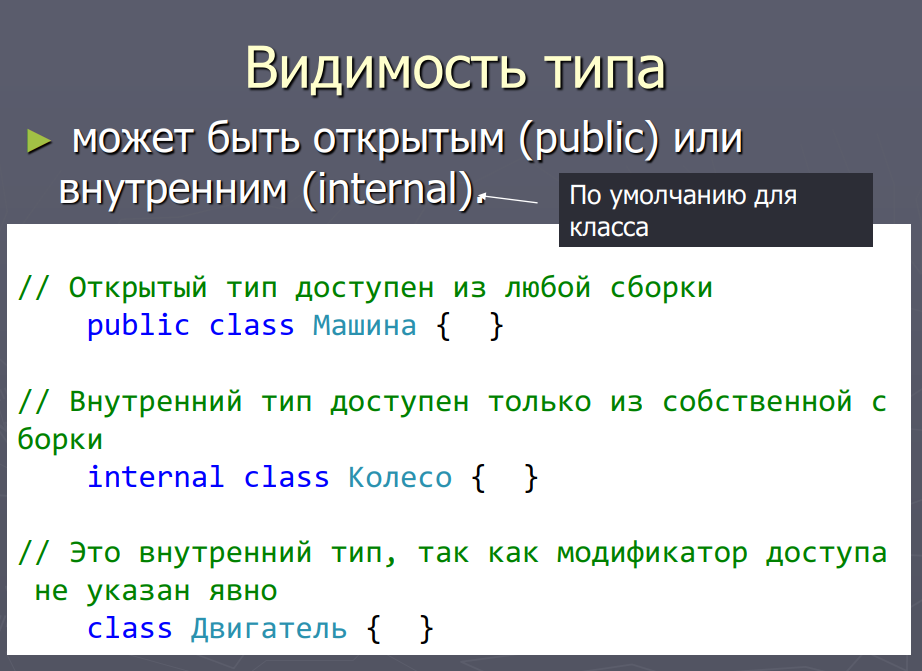






6. Какие ключевые слова можно использовать при определении класса?

*Ответ:* При определении класса используют ключевые слова public/internal (либо без них, тогда подразумевается internal), class.



7. В чем отличие между объектом и классом?

*Ответ:* **Класс** – это абстрактное понятие - шаблон, по которому определяется форма объекта. Класс — это определение общего поведения для объектов, который этот класс порождает.

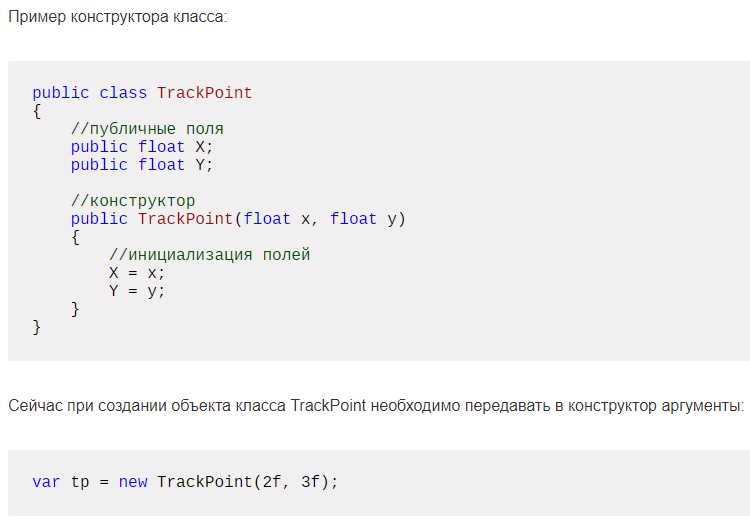
**Объект** – это физическая реализация класса (шаблона).



8. Что такое конструктор? Когда вызывается конструктор?

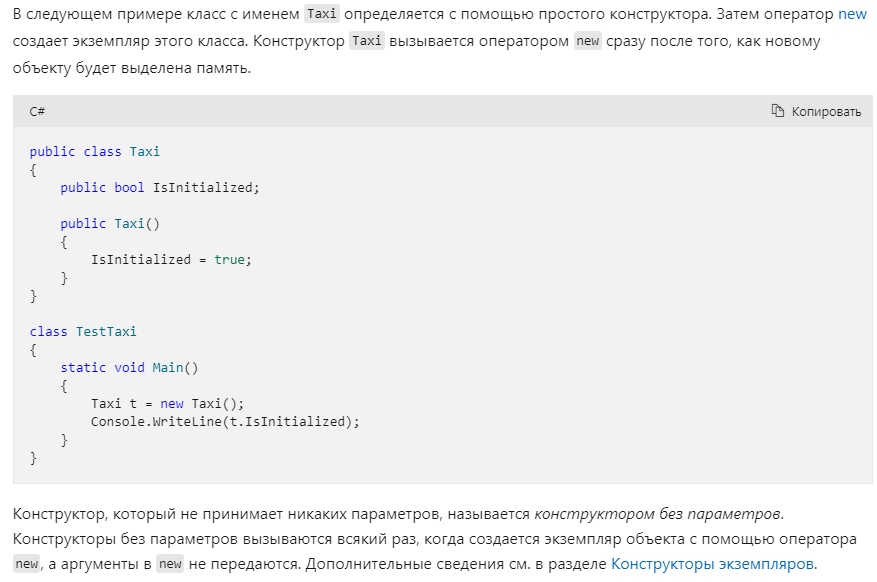
*Ответ:* **Конструктор** – специальный метод который вызывается при создании нового экземпляра класса, он выделяет память необходимую для хранения объекта, и как правило выполняет инициализацию полей и свойств.

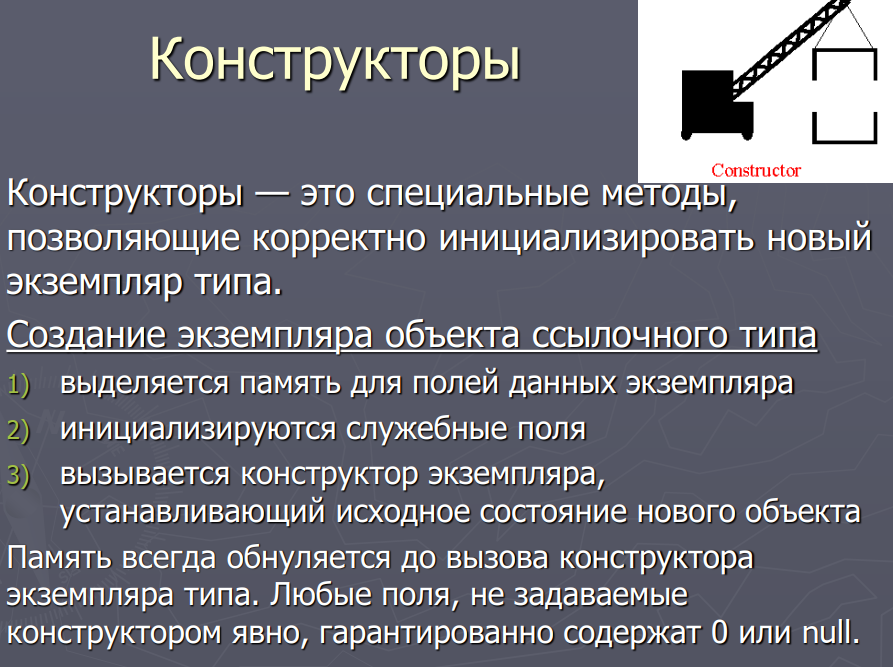
**Имя конструктора должно быть идентично имени класса.** Если в классе не задан конструктор, то компилятор **генерирует конструктор по умолчанию без параметров.**



Как правило, конструктор используется для задания первоначальных значений переменных экземпляра, определенных в классе, или же для **выполнения** любых других установочных **процедур**, которые **требуются для создания полностью сформированного объекта.**

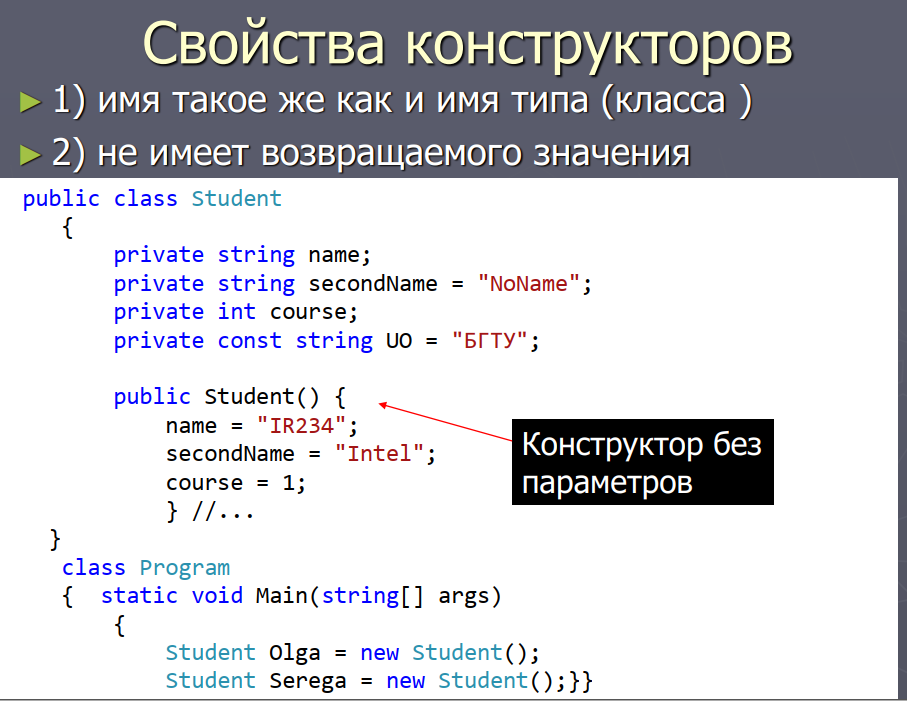
Конструкторы вызываются при создании нового объекта данного класса **с помощью оператора new**. Конструкторы выполняют инициализацию объекта.

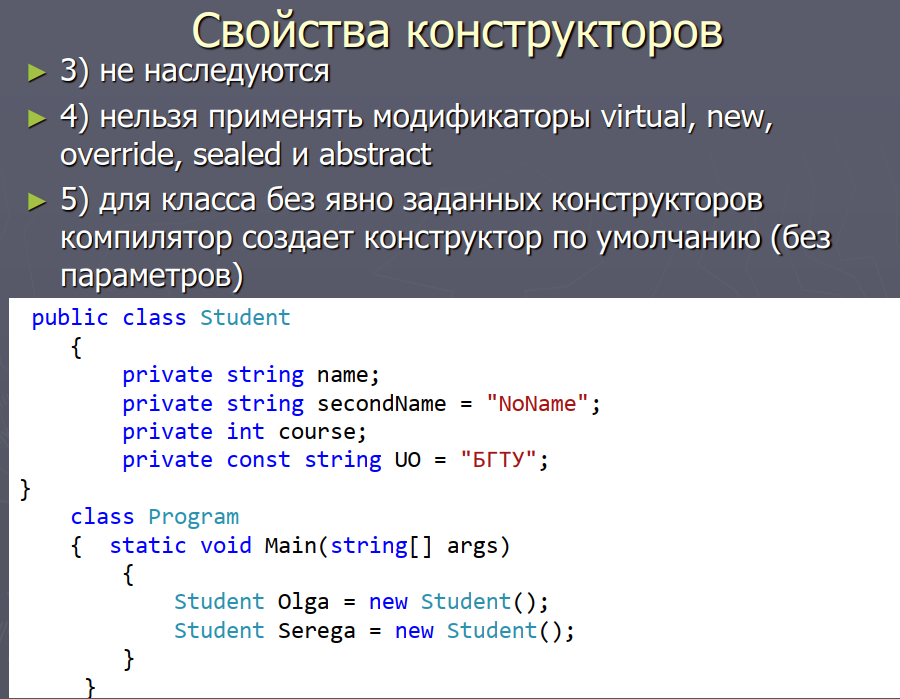


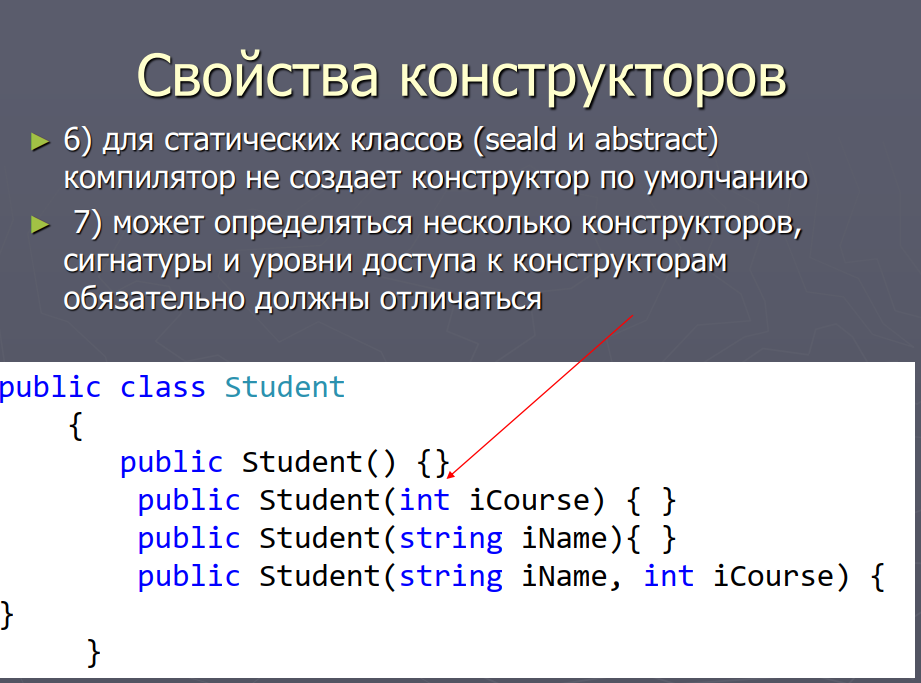


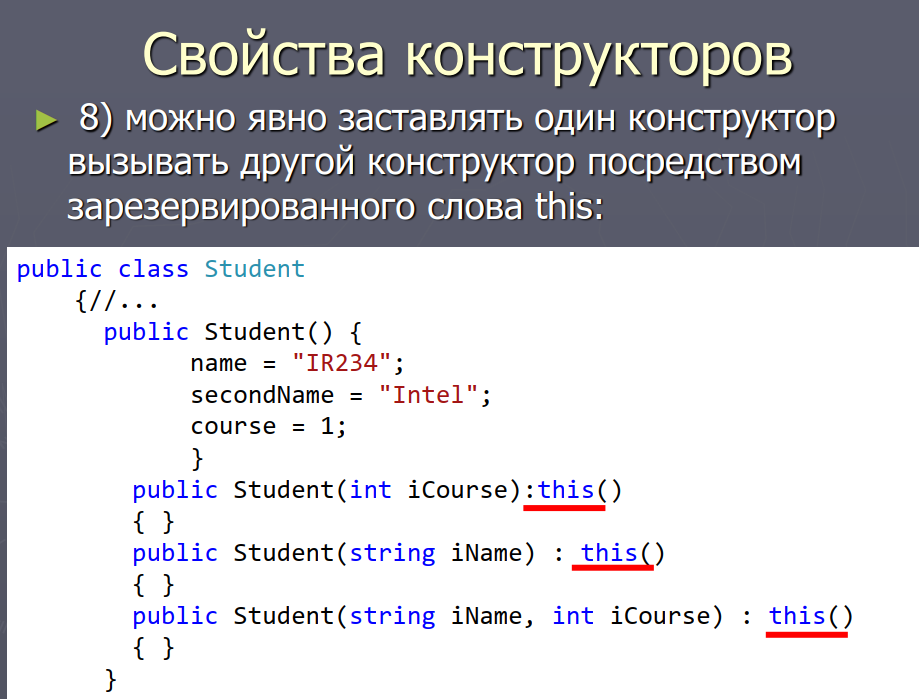
9. Перечислите свойства конструктора?

*Ответ:*









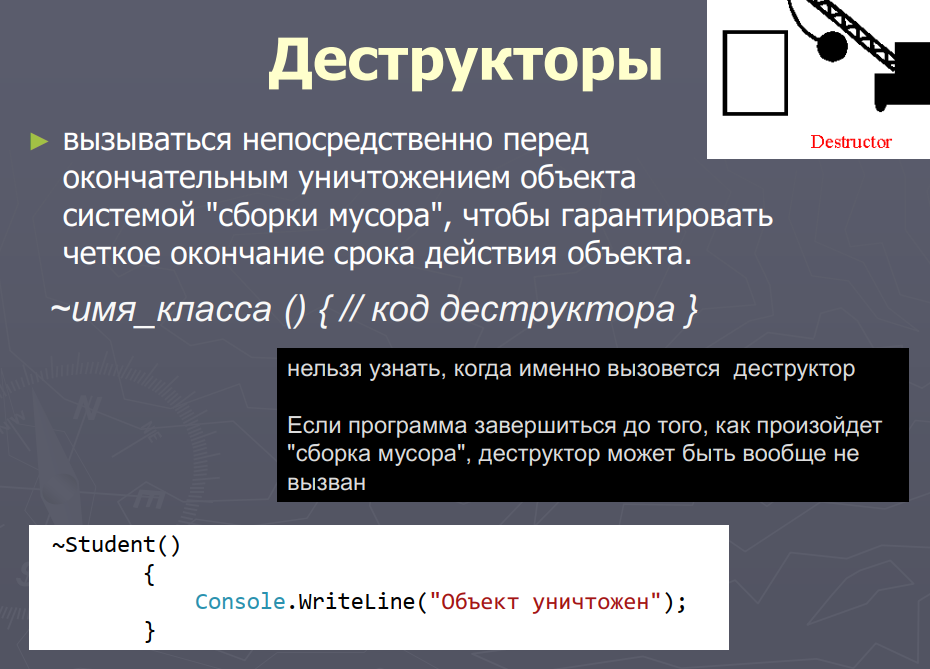
10. Что такое деструктор (destructor) ?

*Ответ:* При использовании оператора **new** свободная память для создаваемых объектов динамически распределяется из доступной буферной области [оперативной памяти](https://professorweb.ru/my/glossariy/glossy/files/101.php#o2). Разумеется, оперативная память не бесконечна, и поэтому свободно доступная память рано или поздно исчерпывается. Это может привести к неудачному выполнению оператора new из-за нехватки свободной памяти для создания требуемого объекта.

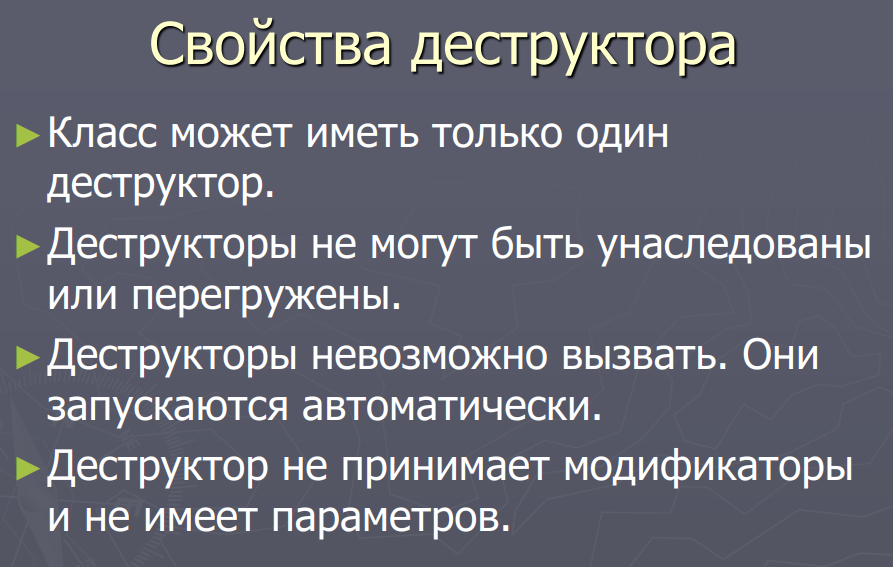
Именно по этой причине **одной из главных функций любой схемы динамического распределения памяти является освобождение памяти от неиспользуемых объектов, чтобы сделать ее доступной для последующего перераспределения.**

**Большинство объектов**, используемых в программах на C#, относятся к **управляемым или managed-коду**, и легко очищаются сборщиком мусора.

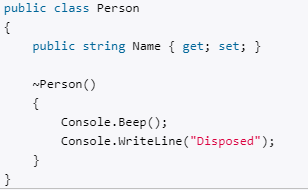
Однако вместе с тем **встречаются также и такие объекты, которые задействуют неуправляемые объекты** (низкоуровневые файловые дескрипторы, сетевые подключения и т.д.). Такие неуправляемые объекты обращаются к API операционной системы через специальные службы. **Сборщик мусора может справиться с управляемыми объектами, однако он не знает, как удалять неуправляемые объекты.** В этом случае разработчик должен сам реализовывать механизмы очистки на уровне программного кода - **деструкторы**.



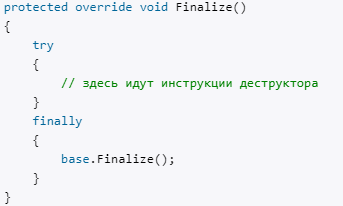
В данном случае в деструкторе в целях демонстрации просто выводится строка на консоль, **но в реальных программах в деструктор вкладывается логика освобождения неуправляемых ресурсов.**



Например, создадим деструктор класса Person:



На деле при очистке сборщик мусора **вызывает не деструктор, а метод Finalize класса Person**. Все потому, что компилятор C# компилирует деструктор в конструкцию, которая эквивалентна следующей:



Метод **Finalize** уже определен в базовом для всех типов классе Object, однако **данный метод нельзя так просто переопределить**. И фактическая его **реализация происходит через создание деструктора**.

11. Что такое this?

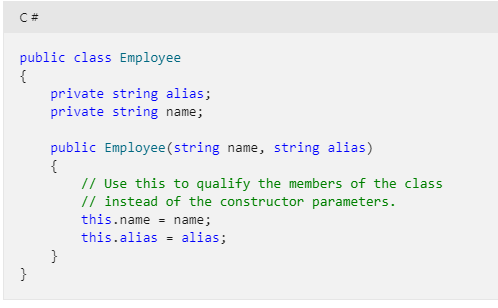
*Ответ:* В методах класса может быть использовано ключевое слово this. По определению **this** – **это неявно присутствующая ссылка на текущий экземпляр класса.**

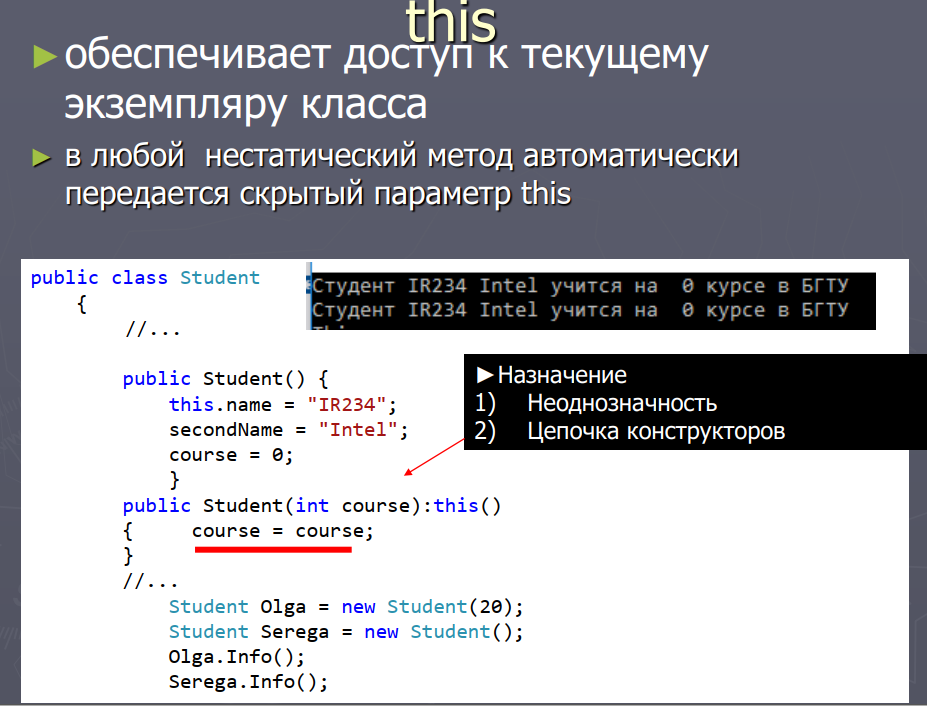
**Ключевое слово this обеспечивает доступ к текущему экземпляру класса.**

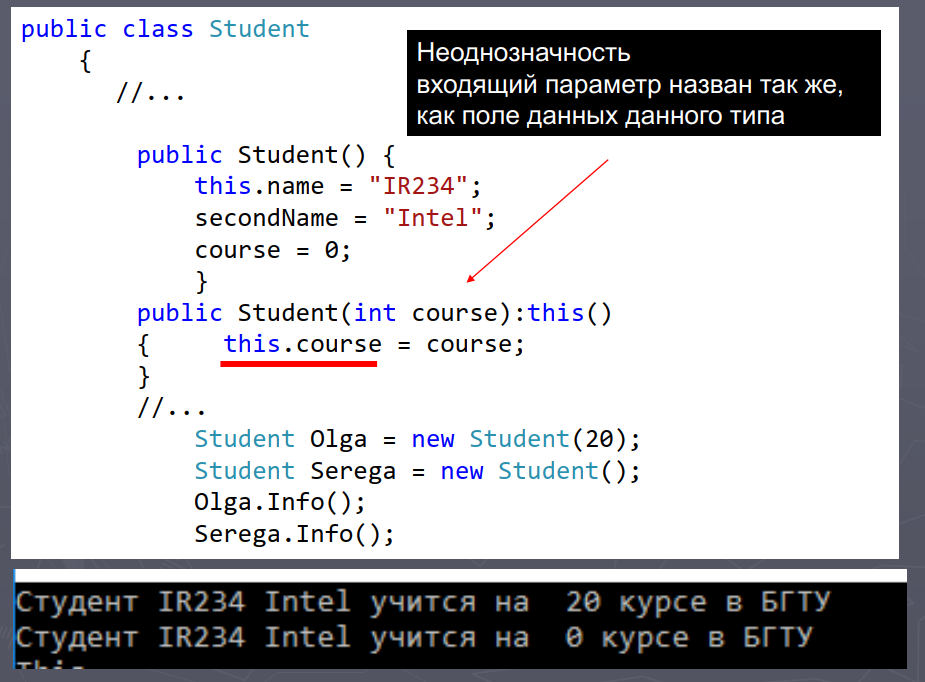
Одно из возможных применений ключевого слова this состоит в том, чтобы разрешать неоднозначность контекста, которая может возникнуть, когда входящий параметр назван так же, как поле данных данного типа.

Ключевое слово this **применяется в следующих случаях**.

1. Когда **в методе класса существует имя, совпадающее с именем поля класса.**
2. Если в классе есть несколько конструкторов, то this применяется для вызова главного конструктора из других (**перенаправление на главный конструктор**). Это осуществляется с целью избежания дублирования кода.
3. **Когда в метод нужно передать ссылку на текущий объект**. В этом случае, при вызове метода, указывается ключевое слово this.



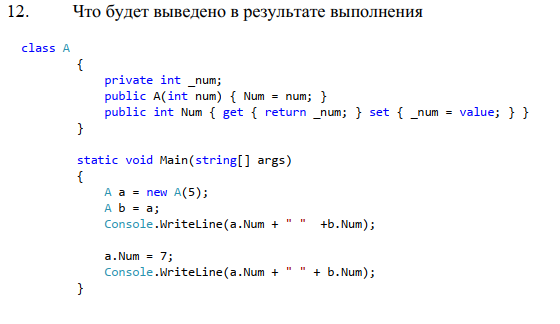


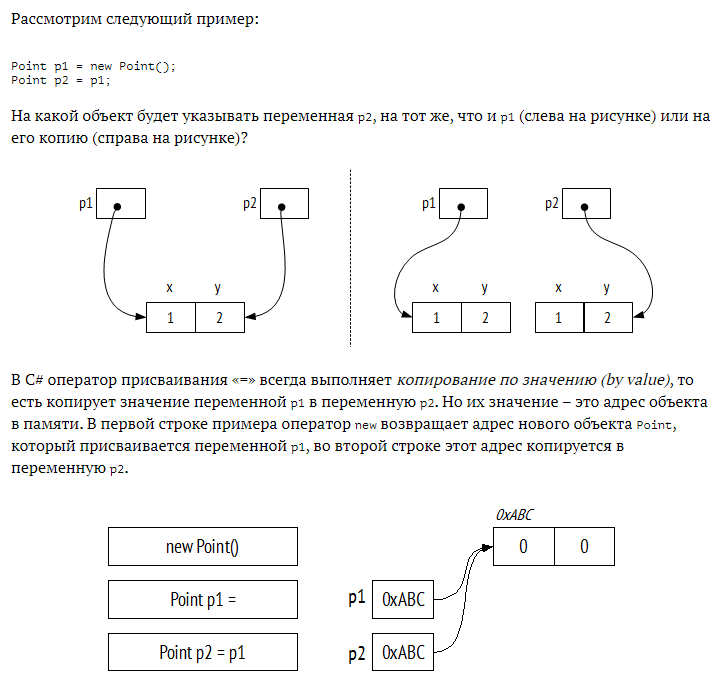


12. Что будет выведено в результате выполнения?

*Ответ:* 5 5

7 7



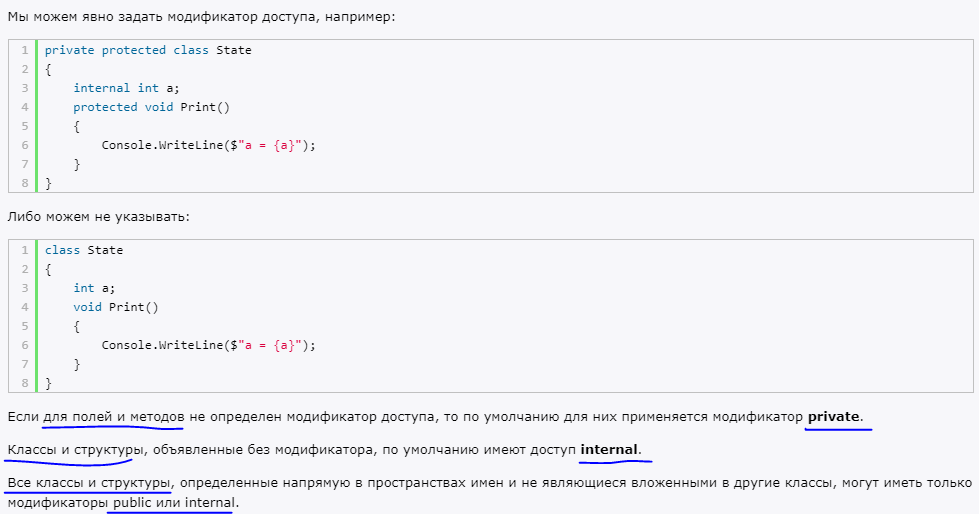
Пояснение: 

13. Какие спецификаторы доступа для класса и методов класса существуют в C#?

*Ответ:* **Все члены класса - поля, методы, свойства - все они имеют модификаторы доступа.** Модификаторы доступа позволяют задать **допустимую область видимости для членов класса**. То есть модификаторы доступа определяют контекст, в котором можно употреблять данную переменную или метод.

В C# применяются следующие модификаторы доступа:

1. **public**: публичный, общедоступный класс или член класса. Такой член **класса доступен из любого места в коде, а также из других программ и сборок.**
2. **private**: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен **только из кода в том же классе** или контексте.
3. **protected**: такой член класса доступен из любого места **в текущем классе или в производных классах**. При этом **производные классы могут располагаться в других сборках.**
4. **internal**: класс и члены класса с подобным модификатором доступны из любого места кода в данной сборки (класс, производный класс и непроизводный класс)
5. **protected internal**: совмещает функционал двух модификаторов. Доступ к полям, свойствам, методам с таким модификатором возможен в классе, в котором они объявлены, в производном/непроизводном классе текущей сборки и в производном классе из другой сборки.
6. **private protected**: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в текущей.





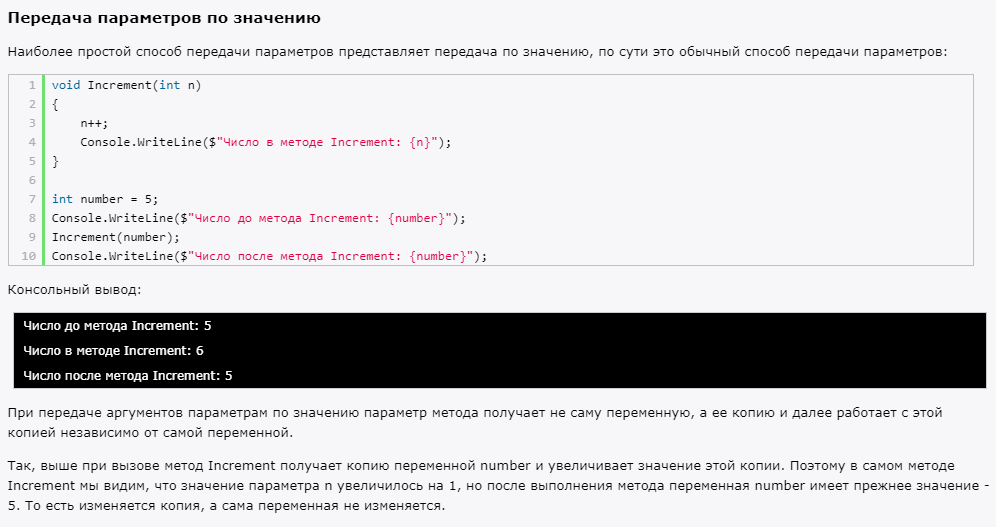
14. Опишите модификатор protected internal.

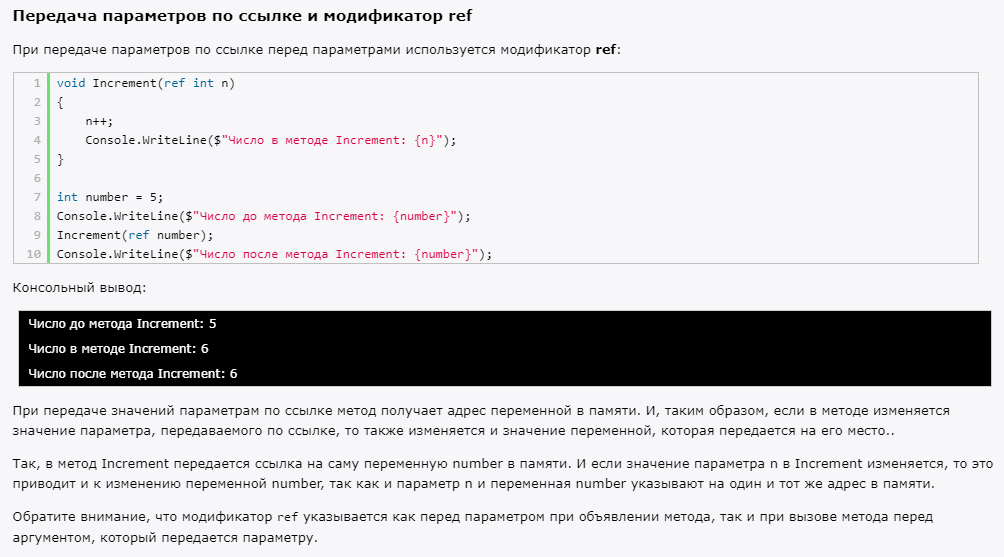
*Ответ*: Поля, методы или свойства с таким методом доступа **доступны в классе, в котором они объявлены, в производном классе и непроизводном классе текущей сборки, в производном классе из другой сборки.**

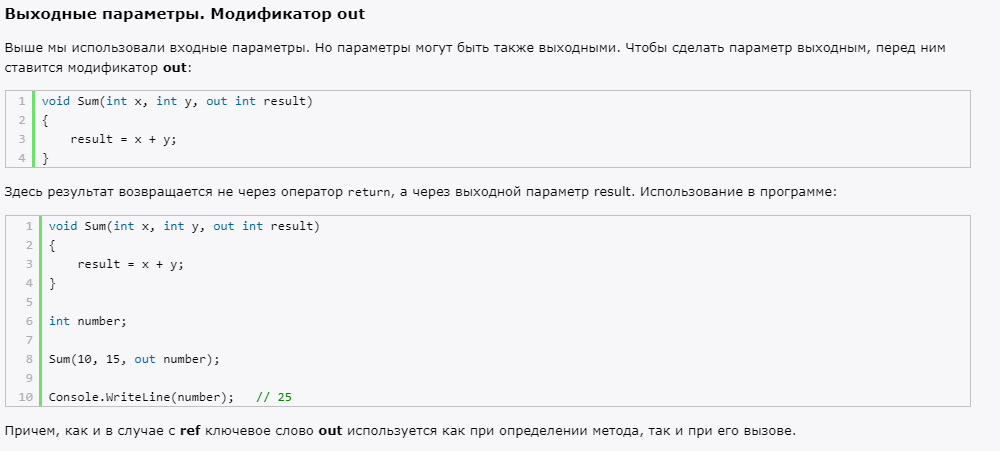


15. Зачем и как используются ref и out параметры функции?

*Ответ:* Существует два способа передачи параметров в метод в языке C#: по **значению** и по **ссылке**.







Прелесть использования подобных выходных параметров состоит в том, что **по сути мы можем вернуть из метода не одно значение, а несколько**.

Использование этих ключевых слов приводит к тому, что **параметр начинает передаваться не по значению, а по ссылке**. Ключевое слово out обозначает почти то же самое, что и ref. **Параметр, перед которым указываются данные операторы, тоже передаётся по ссылке**, т.е. в переменной хранится адрес этого параметра. **В чем разница же между ref и out?** Разница в том, что **out** позволяет вам передать ссылку на переменную **неинициализированную**, а **ref** — только на **инициализированную**.

16. Приведите пример необязательных и именованных параметров метода.

**Параметры позволяют передать в метод некоторые входные данные**. Параметры определяются **через запятую в скобках после названия метода** в виде:

*тип\_метода имя\_метода (тип\_параметра1 параметр1, тип\_параметра2 параметр2, ...)*

*{*

*// действия метода*

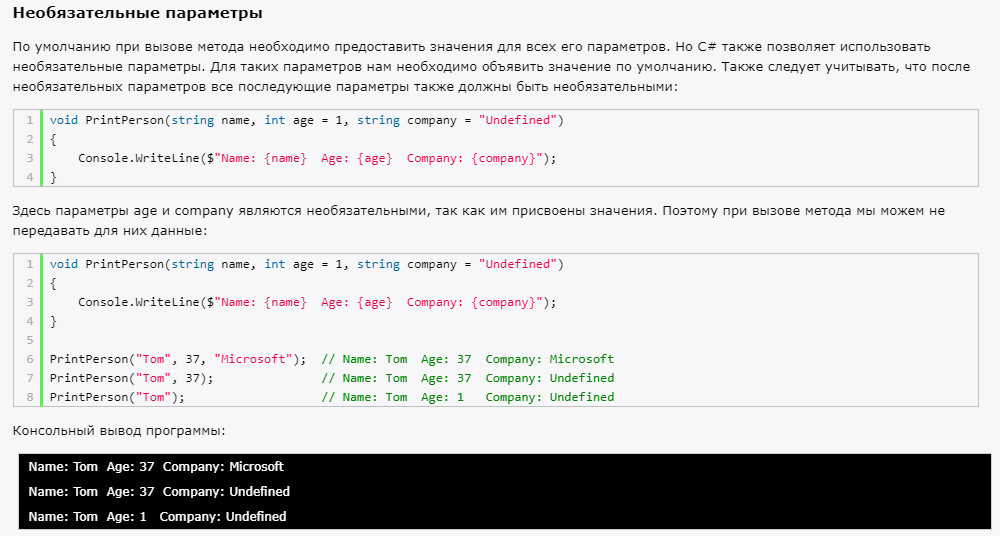
*}*

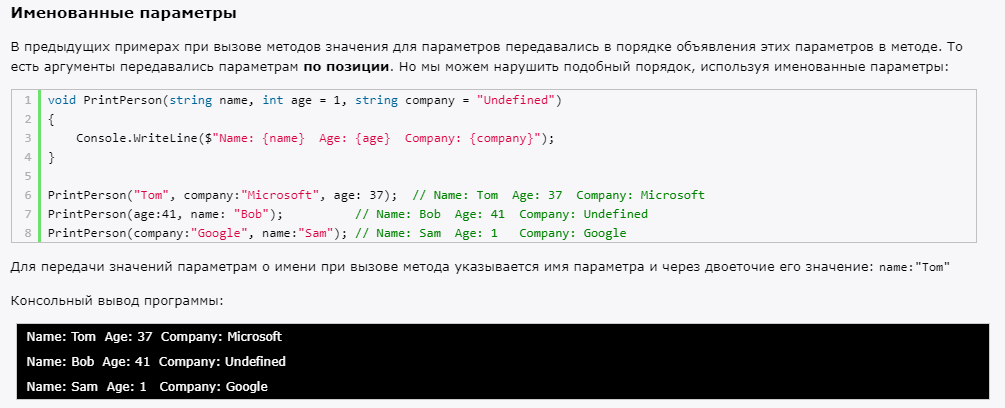
**Определение параметра состоит из двух частей: сначала идет тип параметра и затем его имя.**

Иногда можно встретить такие определения как **формальные параметры** и **фактические параметры**.

**Формальные параметры** - это собственно параметры метода, а **фактические параметры** - значения, которые передаются формальным параметрам. То есть фактические параметры - это и есть аргументы метода.







17. Приведите пример полей класса – статические, константные, только дл  
чтения.

*Ответ:*

**Статические.**

Кроме обычных полей, методов, свойств класс может иметь статические поля, методы, свойства**. Статические поля, методы, свойства относятся ко всему классу и для обращения к подобным членам класса необязательно создавать экземпляр класса. Используется ключевое слово static.**

class Account

{

    public static decimal bonus = 100;

    public decimal totalSum;

    public Account(decimal sum)

    {

        totalSum = sum + bonus;

    }

}

class Program

{

    static void Main(string[] args)

    {

        Console.WriteLine(Account.bonus);      // 100

        Account.bonus += 200;

        Account account1 = new Account(150);

        Console.WriteLine(account1.totalSum);   // 450

        Account account2 = new Account(1000);

        Console.WriteLine(account2.totalSum);   // 1300

        Console.ReadKey();

    }

}

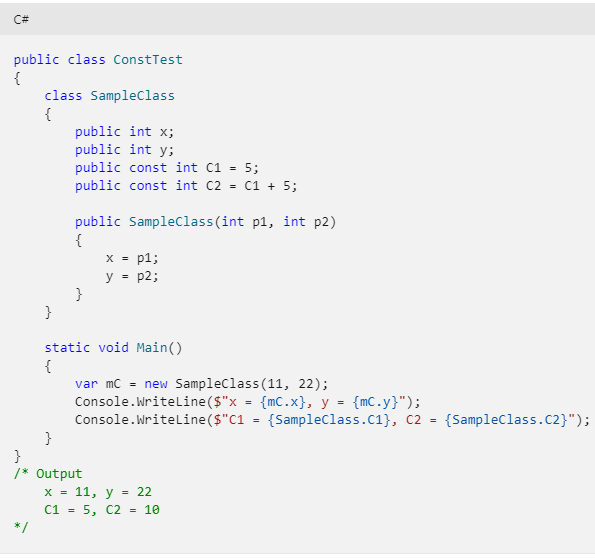
В данном случае класс Account имеет два поля: bonus и totalSum. **Поле bonus является статическим, поэтому оно хранит состояние класса в целом, а не отдельного объекта**. И поэтому мы можем обращаться к этому полю по имени класса.

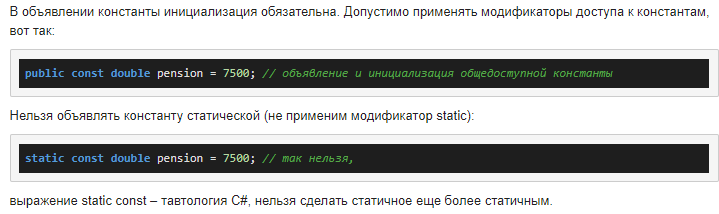
Console.WriteLine(Account.bonus);      // 100

Account.bonus += 200;

**Константные.**

Для объявления константного поля используется ключевое слово **const**.

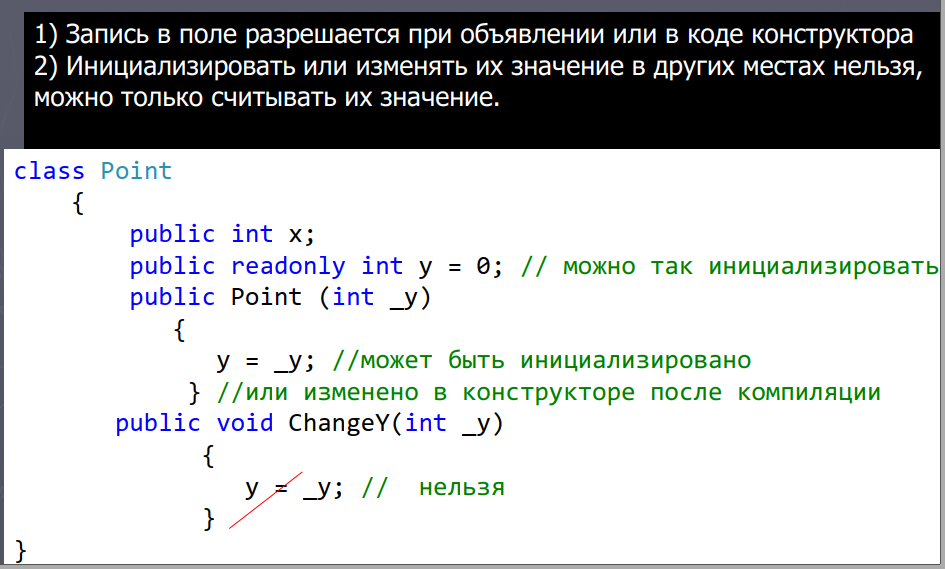




**Только для чтения.**

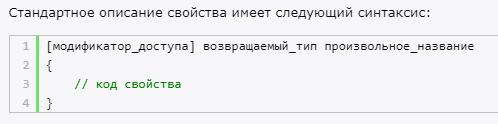
Поля для чтения можно **инициализировать при их объявлении либо на уровне класса**, либо **инициилизировать и изменять в конструкторе**. Инициализировать или изменять их значение в других местах нельзя, **можно только считывать их значение.**

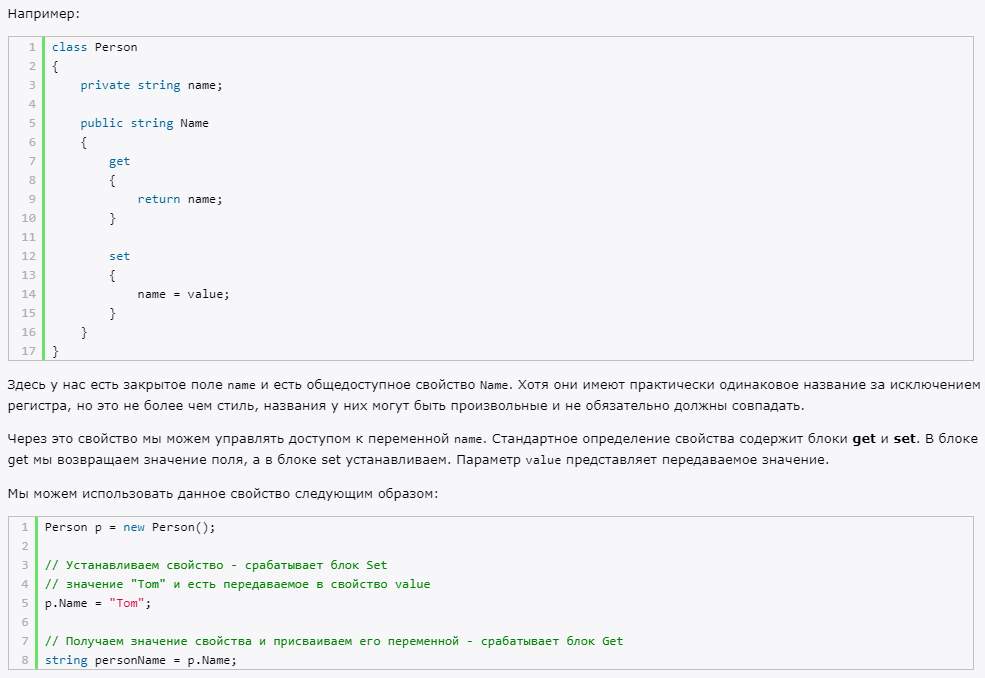
Поле для чтения объявляется с ключевым словом **readonly.**

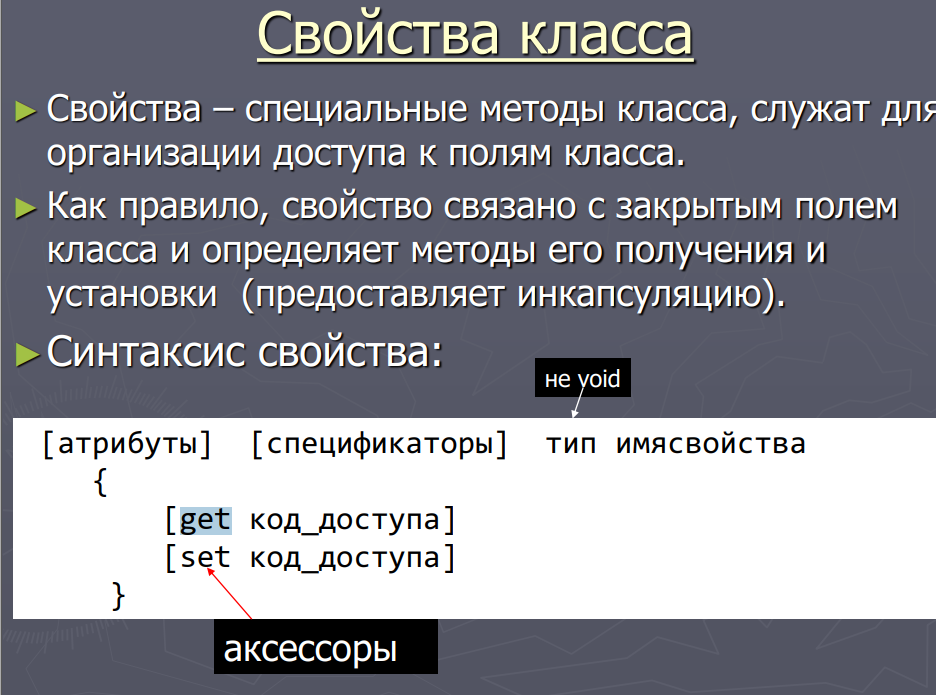


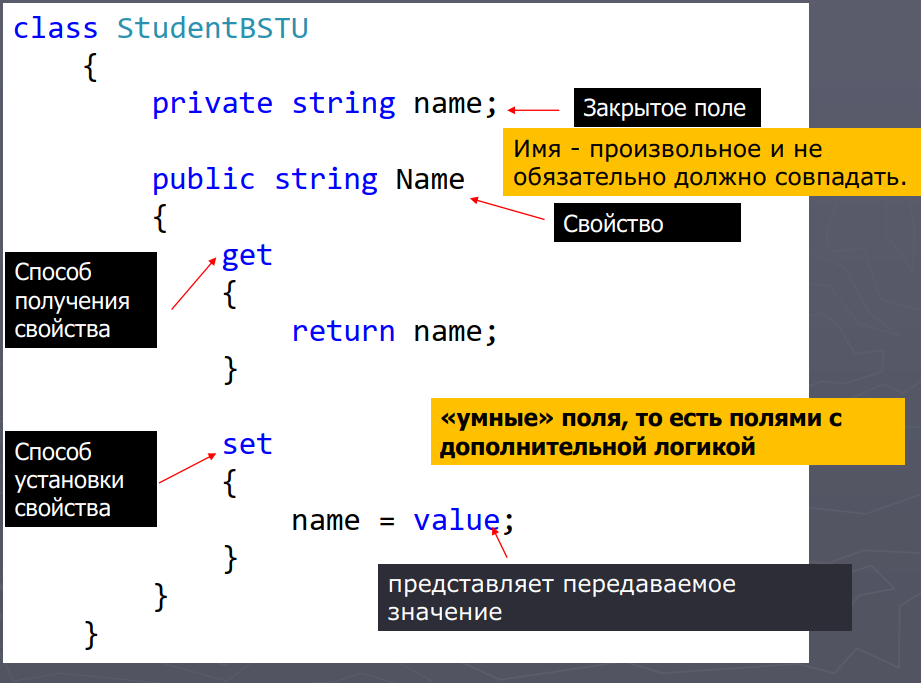
18. Приведите пример определения свойств класса. Как свойства связаны  
инкапсуляцией?

*Ответ:* Кроме обычных методов в языке C# предусмотрены специальные методы доступа, которые называют свойства. **Они обеспечивают простой доступ к полям классов и структур, узнать их значение или выполнить их установку.**







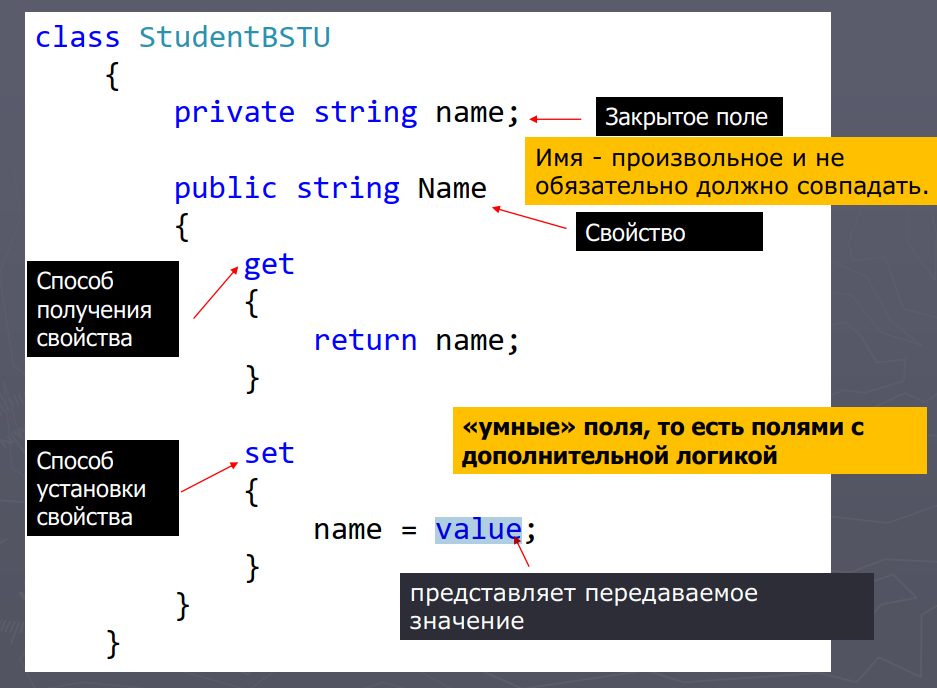


**С идеей инкапсуляции программной логики тесно связана идея защиты данных (модификаторы доступа)**. В идеале **данные состояния объекта должны быть специфицированы с использованием ключевого слова private (или, возможно, protected).** Таким образом, **внешний мир должен вежливо попросить, если захочет изменить или получить лежащее в основе значение**. Это хороший принцип, поскольку общедоступные элементы данных можно легко повредить (даже нечаянно, а не преднамеренно).

Основной единицей инкапсуляции в C# является класс, который определяет форму объекта. Он описывает данные, а также код, который будет ими оперировать. В C# описание класса служит для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

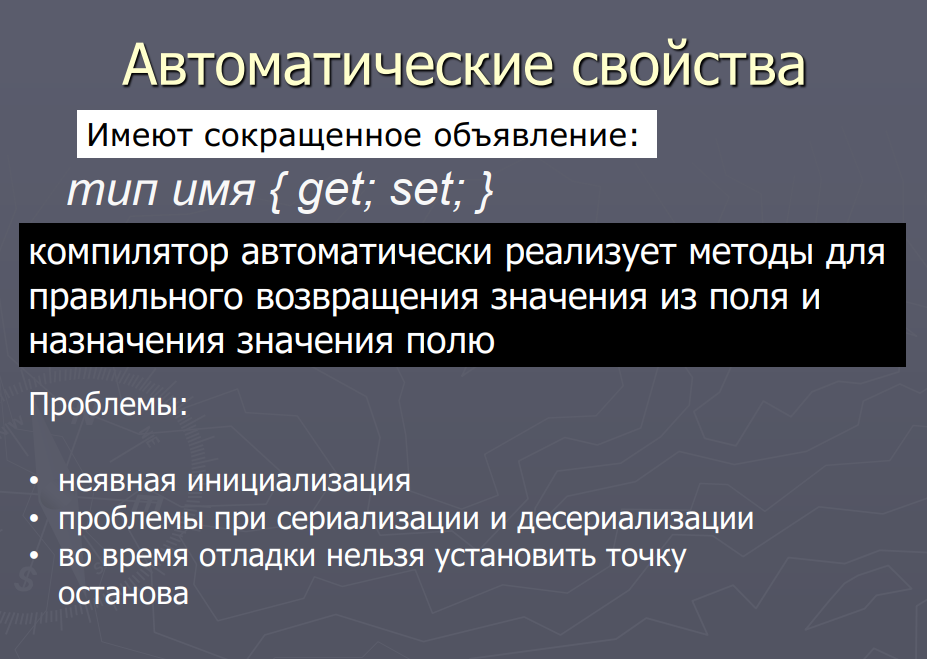
19. Назовите явное имя параметра, передаваемого в метод set свойства класса

*Ответ:* value.

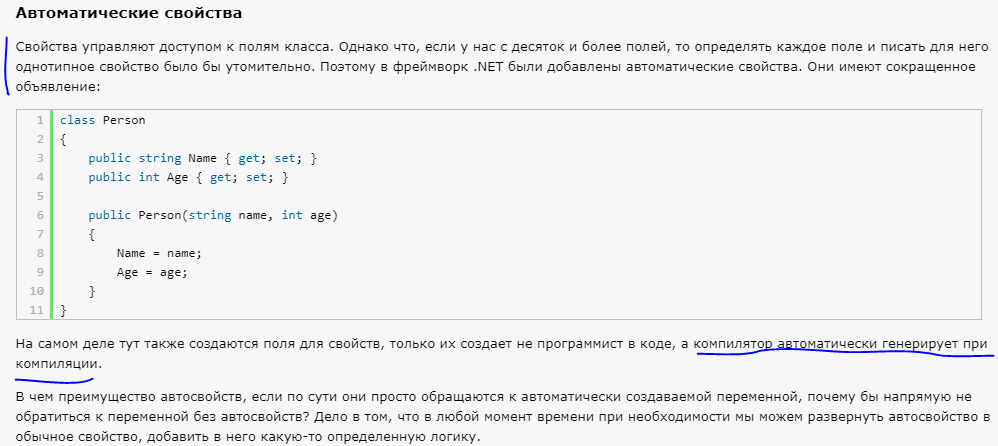


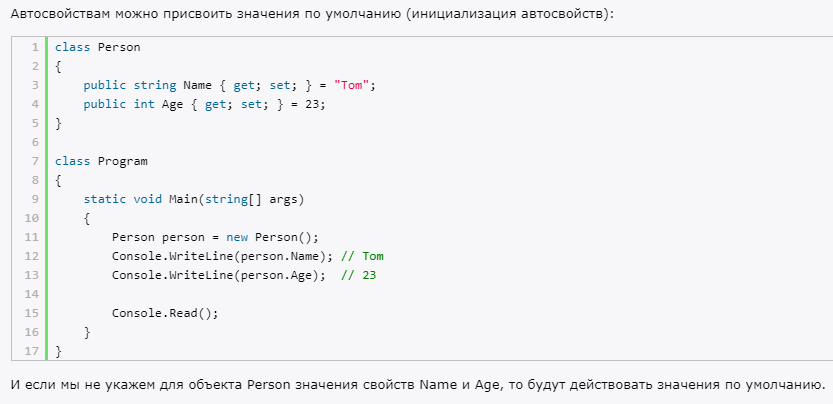
20. Что такое автоматические свойства?

*Ответ:*









21. Что такое индексаторы класса? Какие ограничения существуют н  
индексатор?

*Ответ:* **Индексаторы позволяют индексировать объекты и обращаться к данным по индексу**. Фактически **с** **помощью индексаторов мы можем работать с объектами как с массивами**. По форме они напоминают свойства со стандартными блоками **get и set**, которые возвращают и присваивают значение.

В отличие от свойств индексатор не имеет названия. Вместо него указывается ключевое слово this, после которого в квадратных скобках идут параметры. Индексатор должен иметь как минимум один параметр.

**возвращаемый\_тип this [Тип параметр1, ...]**

**{**

**get { ... }**

**set { ... }**

**}**

Например:

class Person

{

    public string Name { get; set; }

}

class People

{

    Person[] data;

    public People()

    {

        data = new Person[5];

    }

    // индексатор

    public Person this[int index]

    {

        get

        {

            return data[index];

        }

        set

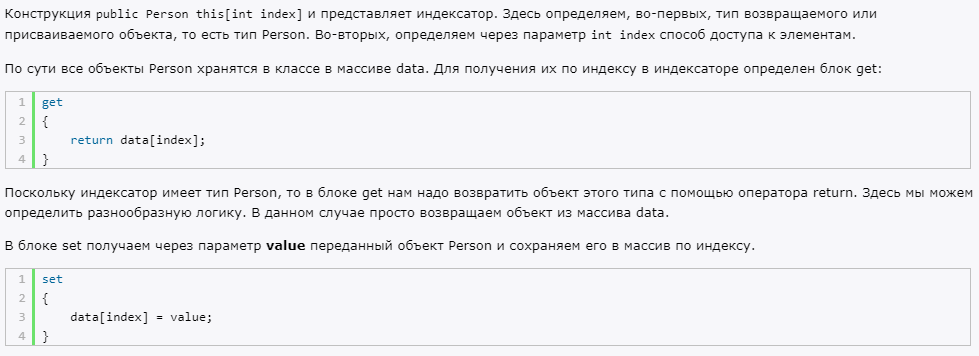
        {

            data[index] = value;

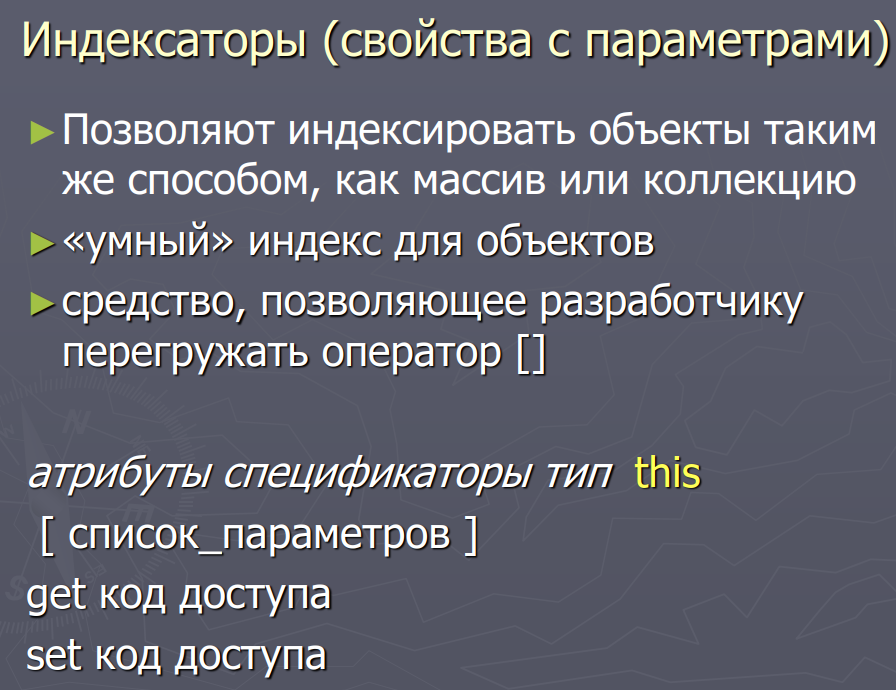
        }

    }

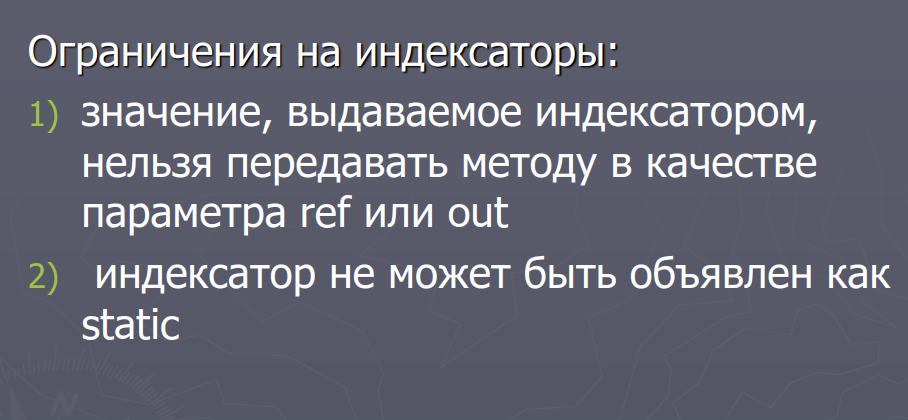
}







Ограничения:





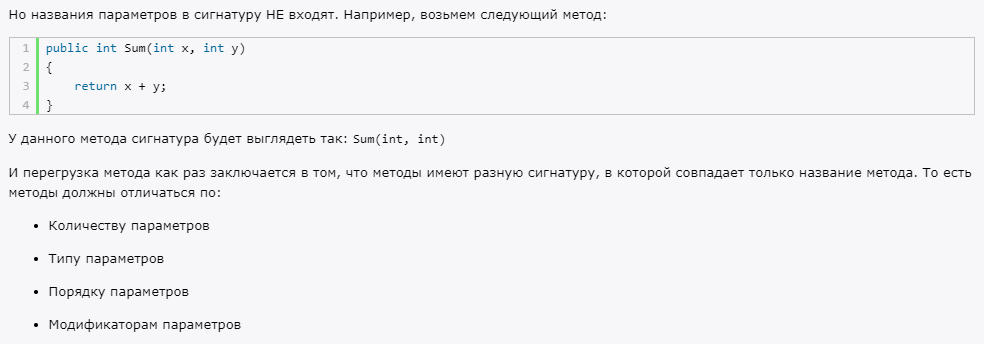
22. Что такое перегруженный метод?

*Ответ:* Иногда возникает необходимость создать **один и тот же метод, но с разным набором параметров**. И в зависимости от имеющихся параметров применять определенную версию метода**. Такая возможность еще называется перегрузкой методов (method overloading).**

И в языке C# мы можем **создавать в классе несколько методов с одним и тем же именем, но разной сигнатурой**. Что такое **сигнатура**? Сигнатура складывается из следующих аспектов:

1. Имя метода
2. Количество параметров
3. Типы параметров
4. Порядок параметров
5. Модификаторы параметров

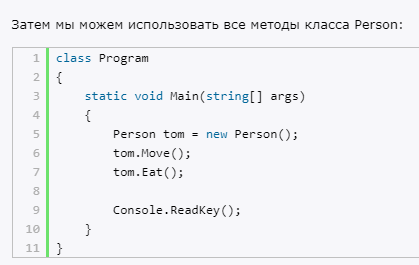
**Но названия параметров в сигнатуру НЕ входят.**



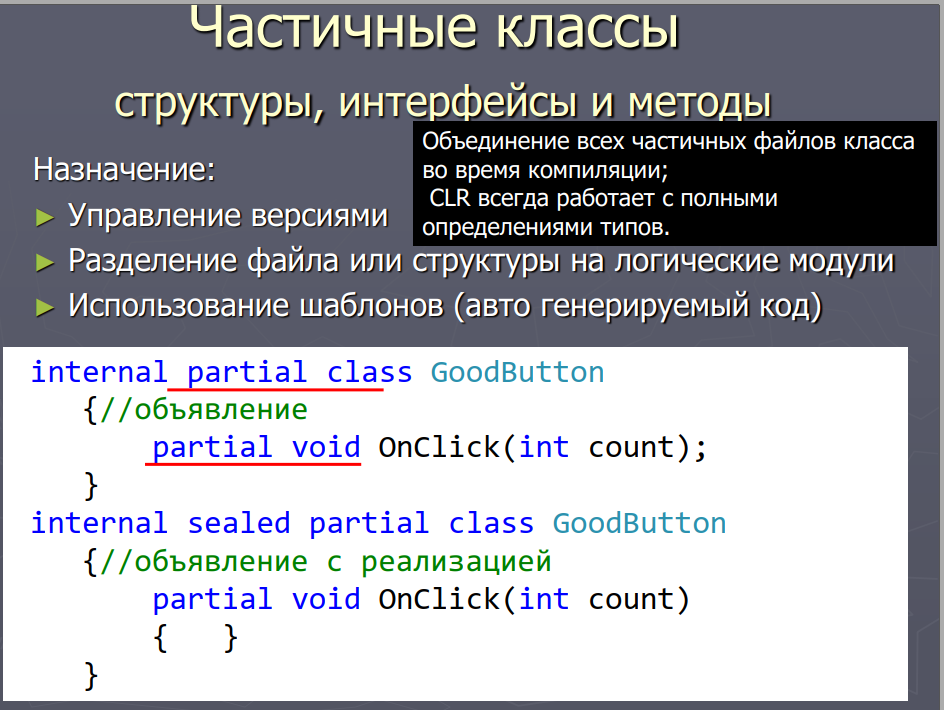
23. Что такое partial класс и какие его преимущества?

*Ответ:* Классы могут быть частичными. То есть мы можем иметь несколько файлов с определением одного и того же класса, и при компиляции все эти определения будут скомпилированы в одно.





Это полезно**, когда у вас есть очень большой класс - вы можете сохранить его в нескольких файлах, чтобы облегчить работу с различными частями классов**. Например, вы можете иметь все свойства в одном файле и все методы в другом файле, имея при этом только один класс.



24. Что такое анонимный тип в C#?

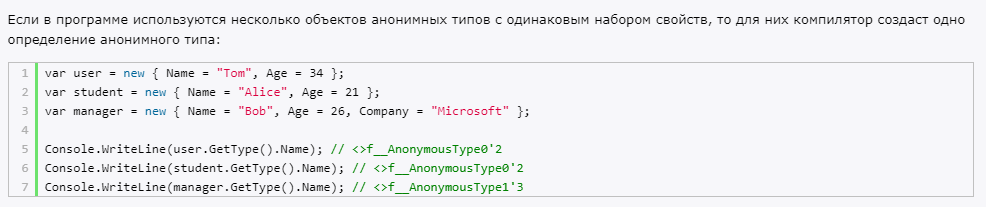
*Ответ:* **Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса**. Анонимный тип определяется с помощью ключевого слова var и инициализатора объектов:

var user = new { Name = "Tom", Age = 34 };

Console.WriteLine(user.Name);

В данном случае user - это объект анонимного типа, у которого определены два свойства Name и Age. И мы также можем использовать его свойства, как и у обычных объектов классов. **Однако тут есть ограничение - свойства анонимных типов доступны только для чтения.**

Для исполняющей среды CLR анонимные типы будут также, как и классы, **представлять ссылочный тип.**



**!!! Следует учитывать, что свойства анонимного объекта доступны для установки только в инициализаторе. Вне инициализатора присвоить им значение мы не можем.**

**Анонимный тип инициализируется с помощью оператора *new* в сочетании с инициализатором объекта** - в этом отношении это очень похоже на создание экземпляра класса, только вы оставляете имя класса. Кроме того, **поскольку за объектом нет класса**, при получении ссылки на объект необходимо **использовать ключевое слово *var*.**

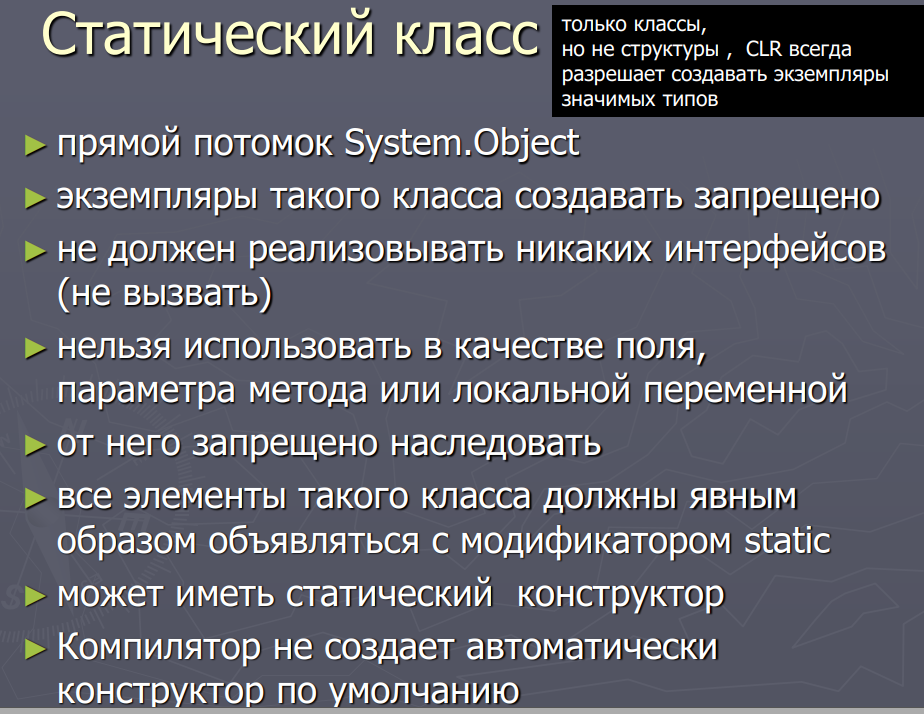
* В отличие от реального класса, анонимный тип не может иметь поля или методы - только свойства.
* После инициализации объекта в него нельзя добавить новые свойства.
* Свойства только для чтения - как только объект был инициализирован, вы не можете изменить их значения.

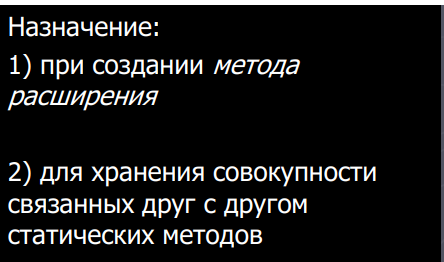
25. Для чего делают статические классы?

*Ответ:* Класс считается статическим, если при его создании, перед ключевым словом **class** указывается ключевое слово **static**. Определение класса статическим, означает то, что этот **класс является набором только статических сущностей** (полей, методов, свойств и т.п.), и, **внимание**, **создавать объекты такого класс нельзя!**

Кроме этого, на статический класс, накладываются как минимум следующие ограничения:

* **статический класс не может служить базовым для других классов**, а также сам **не может быть наследником какого-либо производного класса;**
* класс **не может иметь перегруженных операций.**





Методы расширения (extension methods) **позволяют добавлять новые методы в уже существующие типы без создания нового производного класса**. Эта функциональность бывает особенно полезна, когда нам хочется добавить в некоторый тип новый метод, но сам тип (класс или структуру) мы изменить не можем, поскольку у нас нет доступа к исходному коду.

class Program

{

    static void Main(string[] args)

    {

        string s = "Привет мир";

        char c = 'и';

        int i = s.CharCount(c);

        Console.WriteLine(i);

        Console.Read();

    }

}

public static class StringExtension

{

    public static int CharCount(this string str, char c)

    {

        int counter = 0;

        for (int i = 0; i<str.Length; i++)

        {

            if (str[i] == c)

                counter++;

        }

        return counter;

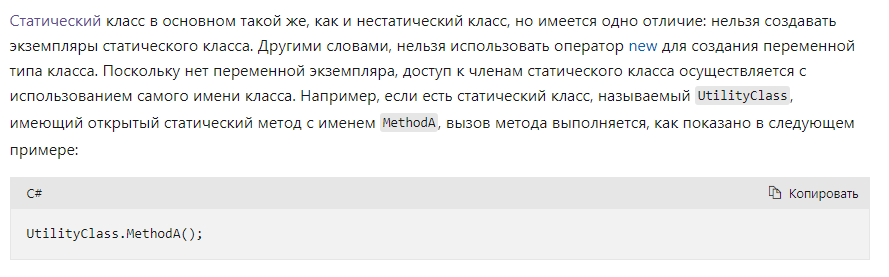
    }

}

**Для того, чтобы создать метод расширения, вначале надо создать статический класс, который и будет содержать этот метод**. В данном случае это класс StringExtension. **Затем объявляем статический метод**. Суть нашего метода расширения - подсчет количества определенных символов в строке.

Собственно, **метод расширения - это обычный статический метод, который в качестве первого параметра всегда принимает такую конструкцию: this имя\_типа название\_параметра**, то есть в нашем случае this string str. Так как наш метод будет относиться к типу string, то мы и используем данный тип.

Применение методов расширения очень удобно, но при этом надо помнить, **что метод расширения никогда не будет вызван, если он имеет ту же сигнатуру, что и метод, изначально определенный в типе.**

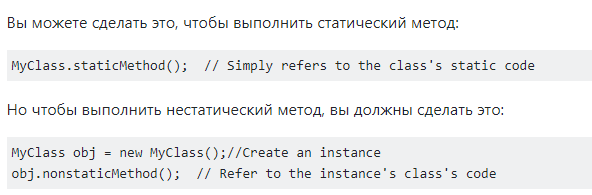


26. В чем отличие статического поля от экземплярного?

*Ответ:*

Члены класса могут быть экземплярными и статическими. Можно понимать статические члены, как принадлежащие классам, а экземплярные члены как принадлежащие объектам.

1. **Статический метод - вызывается на уровне типа** (вызов: Имя\_типа точка Имя\_метода)
2. **Экземплярный метод - вызывается на уровне объявленного экземпляра типа** (вызов: Имя\_переменной точка Имя\_метода).

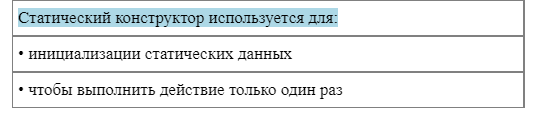


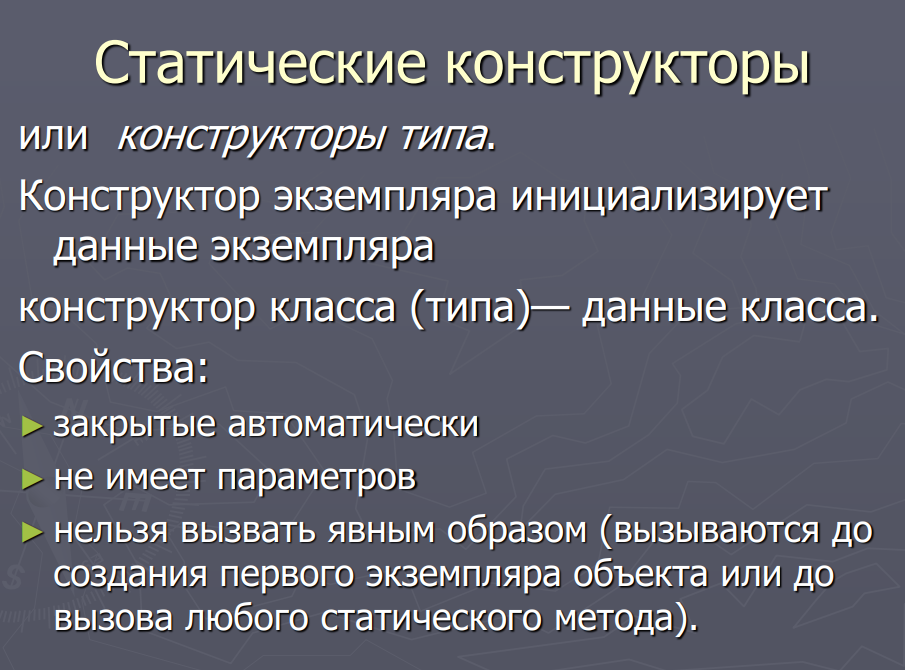
**Метод экземпляра** - это методы, которые требуют создания объекта его класса перед его вызовом.

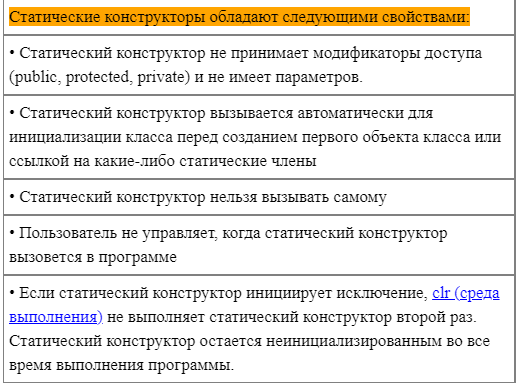
**Статические методы** - это методы, которые можно вызывать без создания объекта класса.

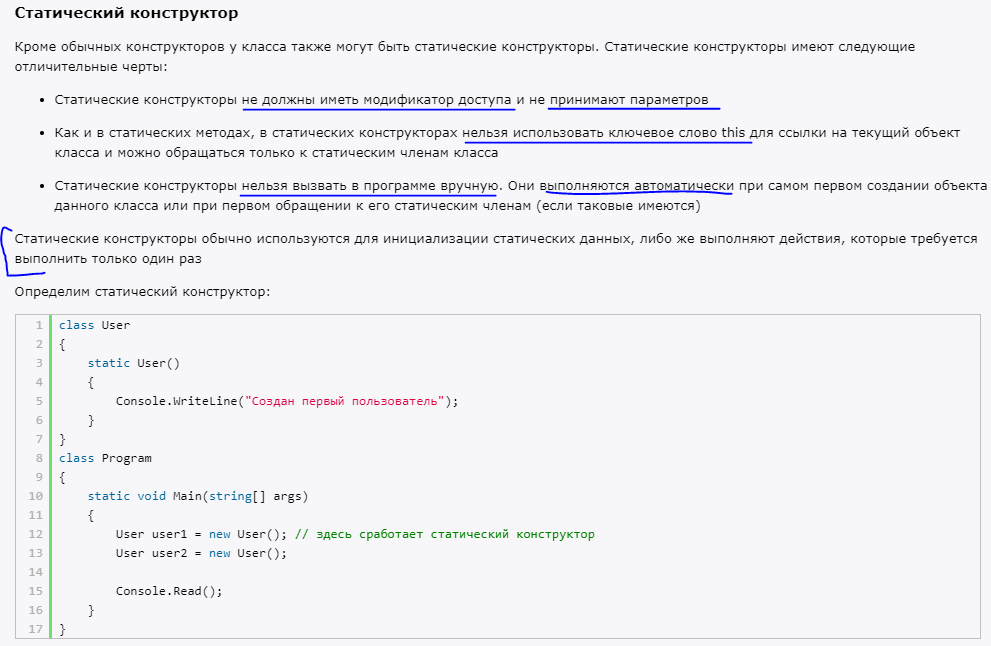
27. Поясните работу статических конструкторов.

*Ответ:*



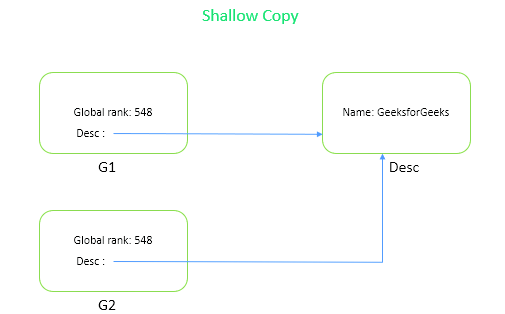
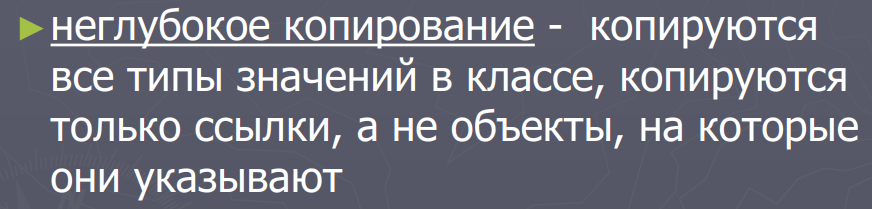


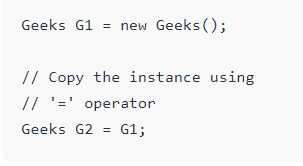


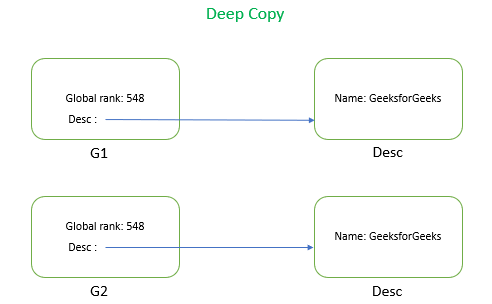


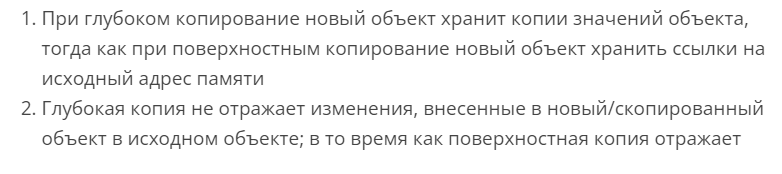
28. Какая разница между поверхностным (shallow) и глубоким (deep  
копированием?

*Ответ:* В общем, когда мы пытаемся скопировать один объект в другой, оба объекта будут использовать один и тот же адрес памяти.

**Неглубокая копия:** создание нового объекта и последующее копирование полей *типа значения* текущего объекта в новый объект. Но когда данные *относятся к ссылочному типу* , то копируется единственная ссылка, но не сам упомянутый объект. Следовательно, оригинал и клон относятся к одному и тому же объекту. 



**Глубокая копия:** это процесс создания нового объекта с последующим копированием полей текущего объекта во вновь созданный объект для создания полной копии внутренних ссылочных типов. Если указанное поле является типом значения, то будет выполнено побитовое копирование поля. Если указанное поле является ссылочным типом, то создается новая копия указанного объекта. 



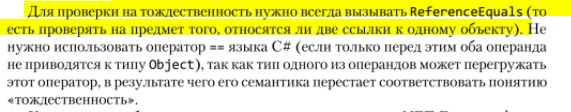
29. В чем разница между равенством и тождеством объектов?

*Ответ:*

**Тождество (идентичность):** переменная содержит тот же экземпляр, что и другая переменная. Тождественность означает, что это один и тот же экземпляр объекта:

Integer a = new Integer(1);

Integer b = a;



**Равенство:** два различных объекта могут использоваться взаимозаменяемо. у них часто один и тот же идентификатор.

Integer c = new Integer(1);

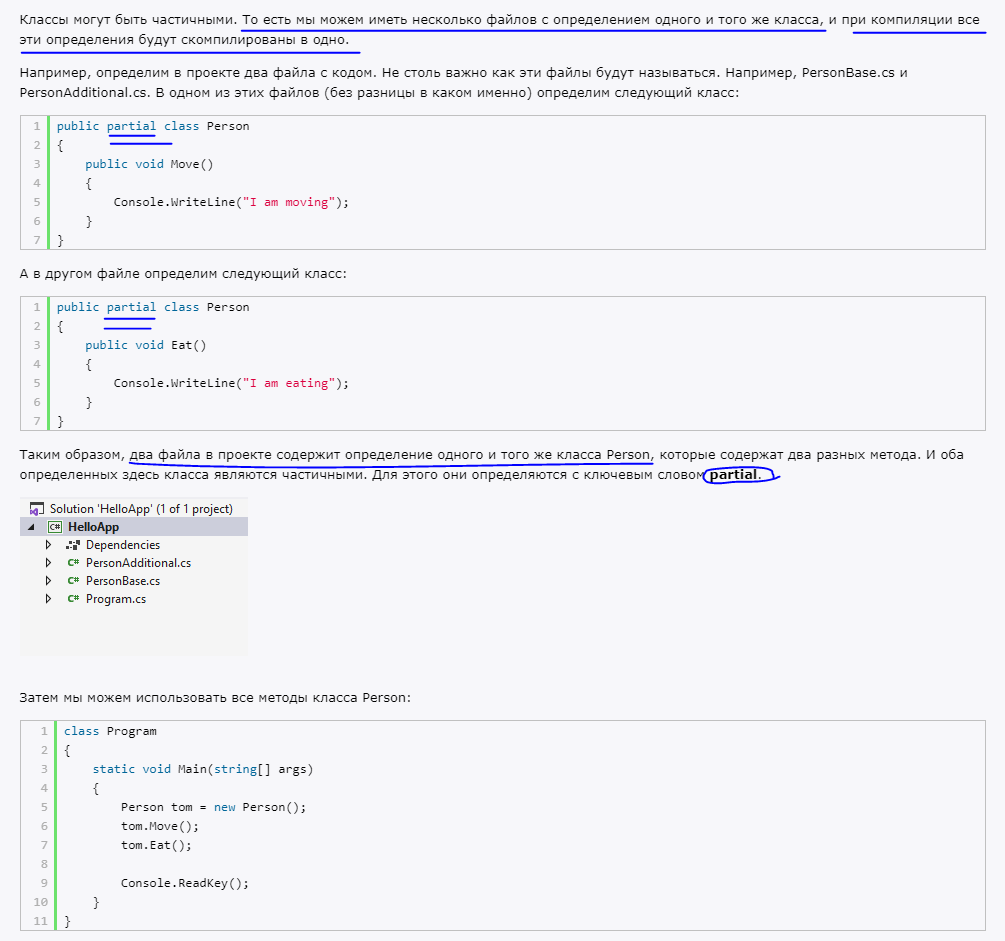
Integer d = new Integer(1);

c равно, но не тождественно d.

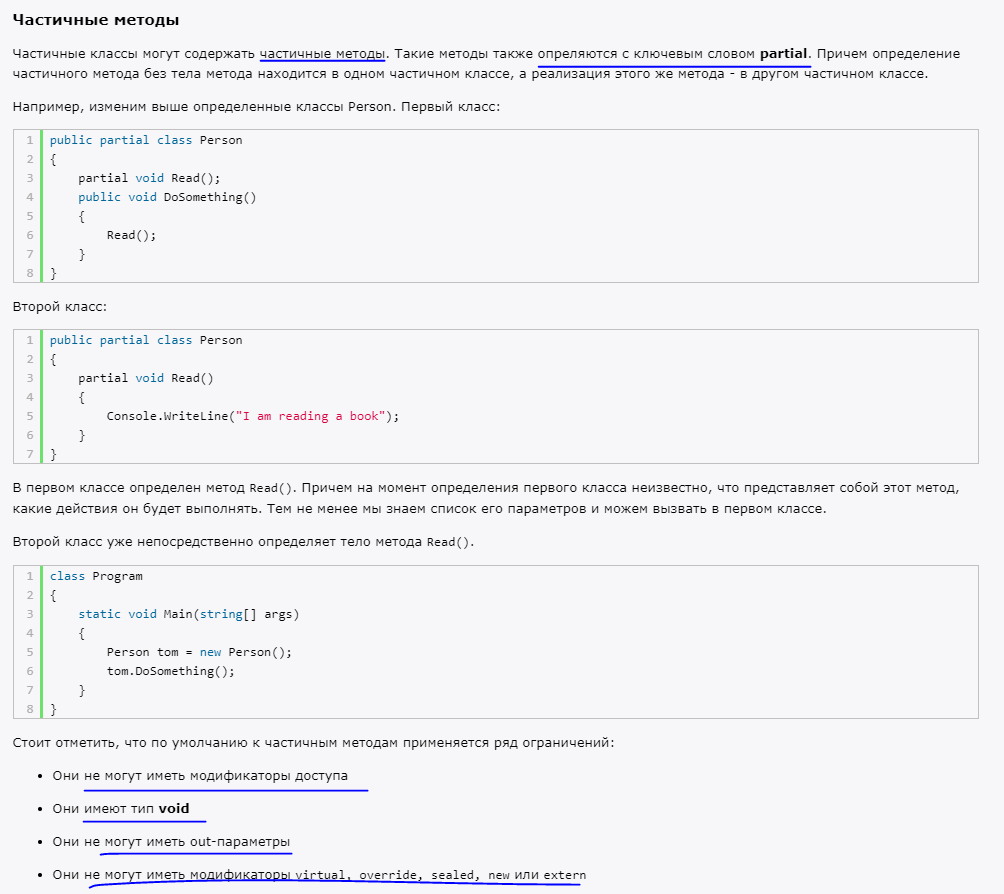
Конечно, две одинаковые переменные всегда равны.

30. Что такое частичные классы и частичные методы?

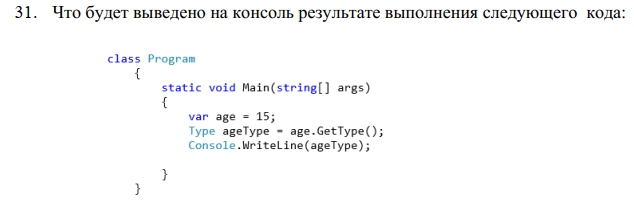
*Ответ:* В языке C# возможно **разбиение определения класса, структуры или интерфейса между двумя или больше исходными файлами**. Каждый исходный файл содержит свою часть определения класса и все такие части собираются во время компиляции.



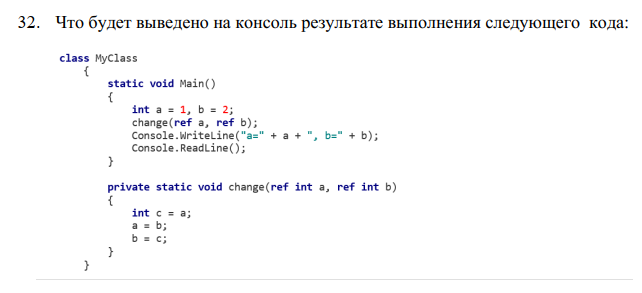
Это полезно**, когда у вас есть очень большой класс - вы можете сохранить его в нескольких файлах, чтобы облегчить работу с различными частями классов**. Например, вы можете иметь все свойства в одном файле и все методы в другом файле, имея при этом только один класс.



31. Что будет выведено на консоль в результате выполнения следующего кода?

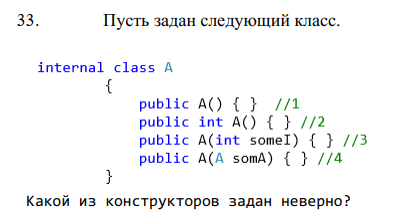


*Ответ:* System.Int32. Метод GetType() возвращает тип текущего экземпляра.

32. Что будет выведено на консоль в результате выполнения следующего кода? 

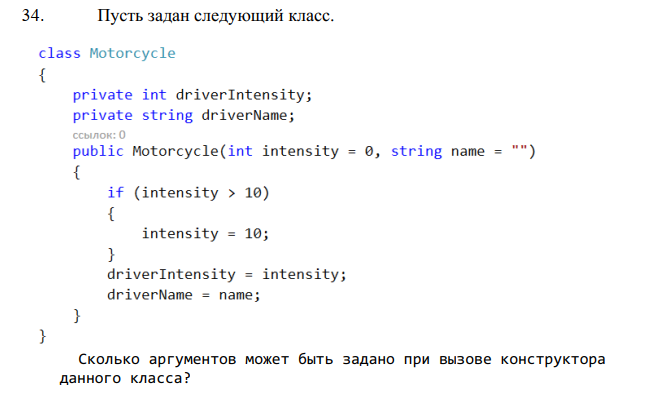
*Ответ:* **a=2, b=1**. Пояснение: **мы передаем параметры по ссылке** (модификатор ref). В метод передались ссылки на переменные, и в методе их значение меняется, это приводит к изменению самих переменных, так как и параметр, и переменная **указывают на один и тот же адрес в памяти**.

33. Пусть задан следующий класс. Какой из конструкторов задан неверно?



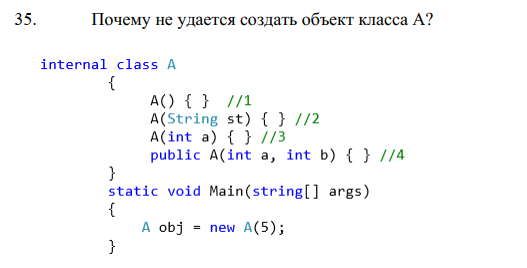
*Ответ:* **2 конструктор задан неверно**, поскольку **конструктор не может иметь возвращаемого значения**. После модификатора доступа сразу идет имя конструктора, которое совпадает с именем его класса.

34. Пусть задан следующий класс. Сколько аргументов может быть задано при вызове конструктора данного класса?

**

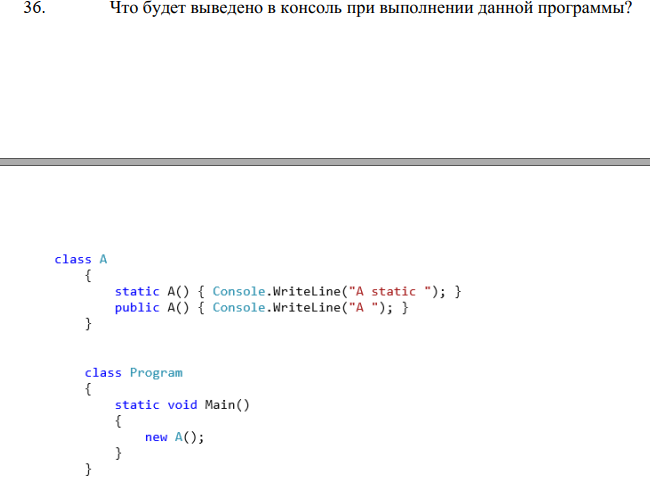
*Ответ:* **Можно не передавать параметры вообще**, поскольку оба параметра, присутствующих в конструкторе, являются **необязательными**, для этих параметров заданы значения по умолчанию, и, если мы не передадим в конструктор никаких значений, будут взяты те, что уже определены по умолчанию. **Можно задать только один или задать два, то есть переопределить параметр/параметры по умолчанию.**

35. Почему не удается создать объект класса А?



*Ответ:* Ошибка в 3 строке. По умолчанию у метода (конструктор – это метод) модификатор доступа private — объявляет метод или свойство доступным только в том классе в котором он присутствует.

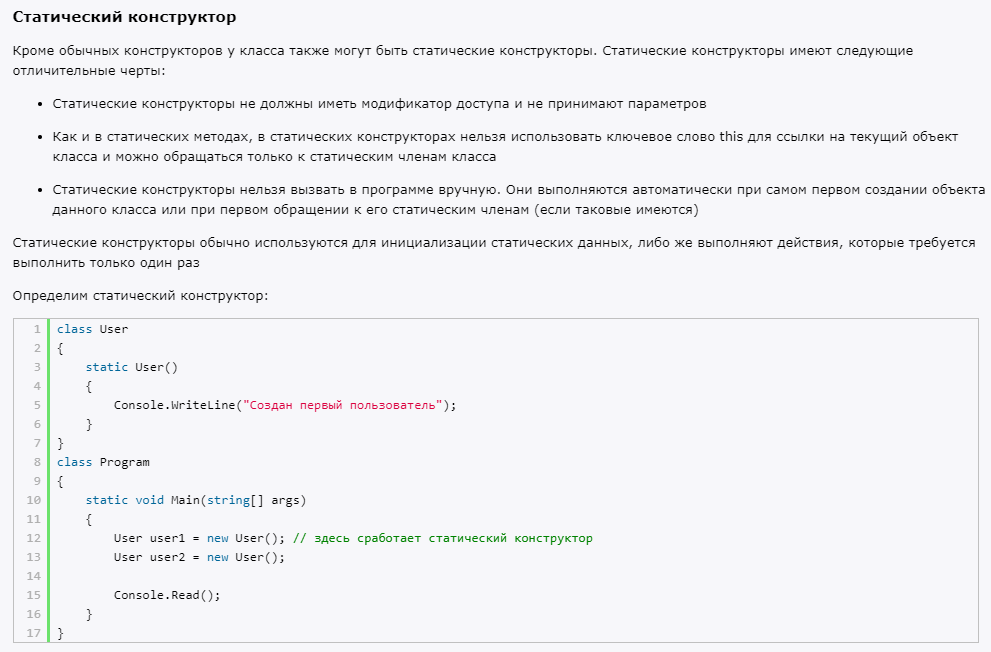
36. Что будет выведено на консоль при выполнении данной программы?



*Ответ:* Будет выведено:  
A static

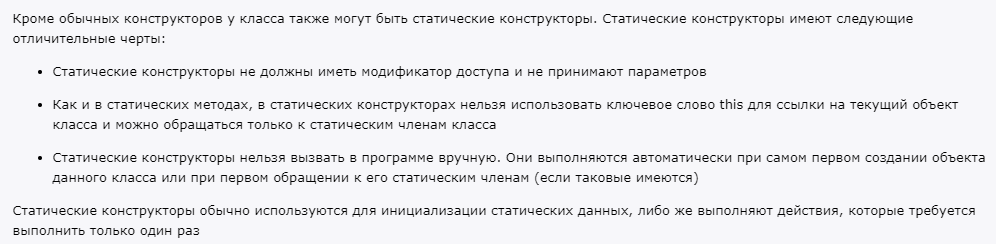
A

Почему? **Первым вызывается статический конструктор static A().** **Использование статического конструктора полностью ложиться на виртуальные плечи среды выполнения.** **Статический конструктор выполняется автоматически при самом первом создании объекта данного класса** или при первом обращении к его статическим членам (если таковые имеются). Затем выполняется конструктор public A().



37. Какая строка приведенного далее класса вызовет ошибку компиляции?

*Ответ:* **Строка 3**, поскольку **значение переменной static readonly может быть изменено только в статическом конструкторе, а конструктор public Points() не является статическим.**



Другие строки не подходят, так как 2 строка – поля для чтения (readonly) можно **инициализировать** при их объявлении либо на уровне **класса**, либо **инициилизировать** и **изменять в конструкторе**. Мы изменяем строку в конструкторе, выполнение успешно.

В 1 строке – задаем новое значение для статического поля, которое относится ко всему классу и для обращения к нему не обязательно создавать экземпляр класса.

