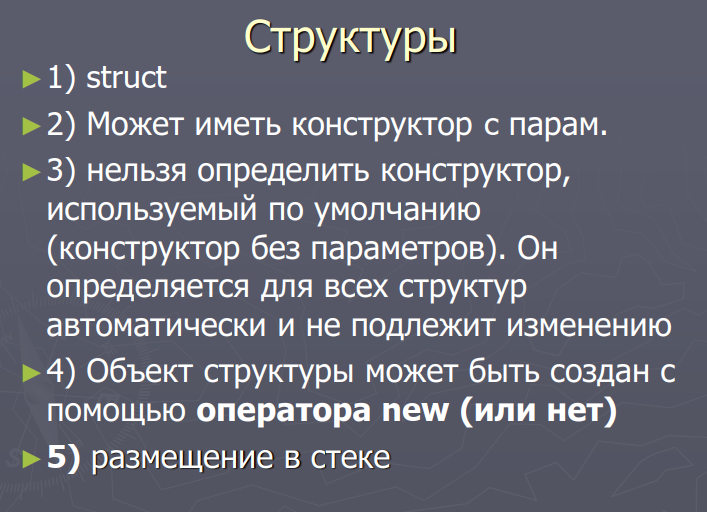
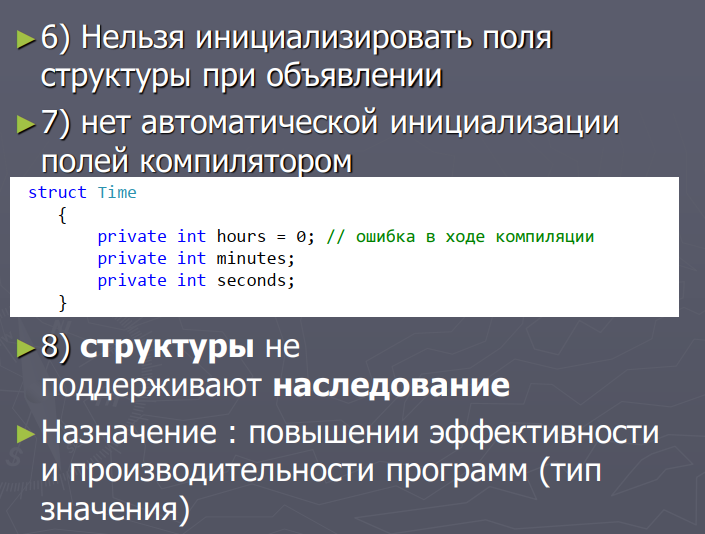
1. Чем класс отличается от структуры?

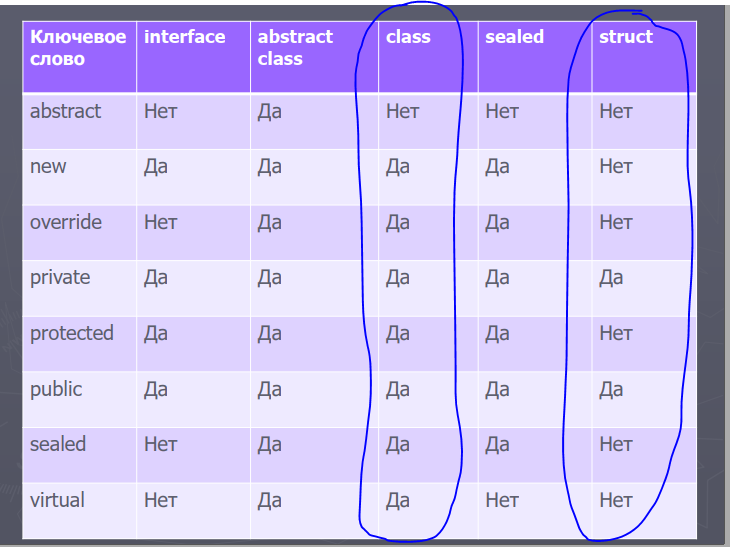
*Ответ:* Структуры синтаксически очень похожи на классы, но существует принципиальное отличие, которое заключается в том, что **класс – является ссылочным типом** (reference type), а **структуры – значимый тип** (value type).

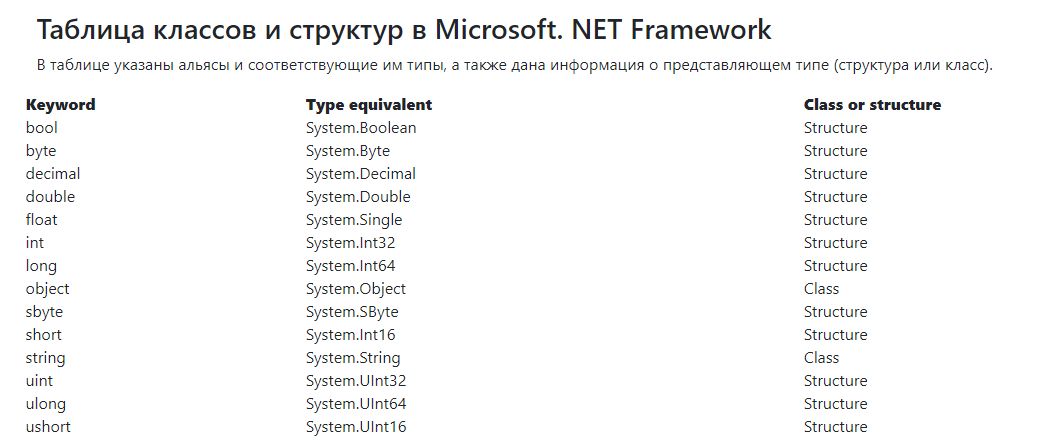
**Поля структуры нельзя инициализировать при объявлении**, поля **класса** – **можно**.   
**Структуры не поддерживают наследование**, **классы** поддерживают **наследование** (но **не множественное**).

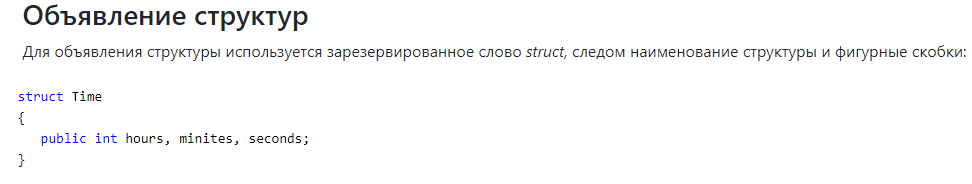
Так же, как и классы, структуры могут иметь поля, методы и конструкторы**. Однако, если в структуре определяется конструктор, то в нем обязательно надо инициализировать все поля структуры.**

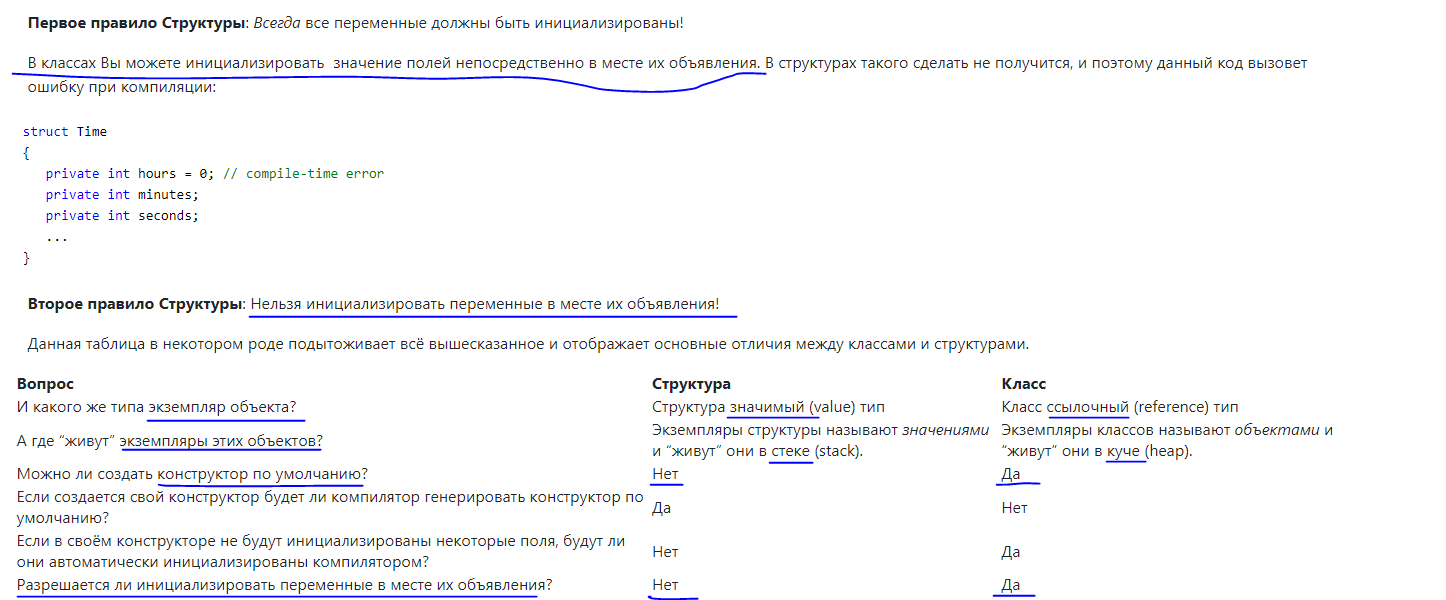




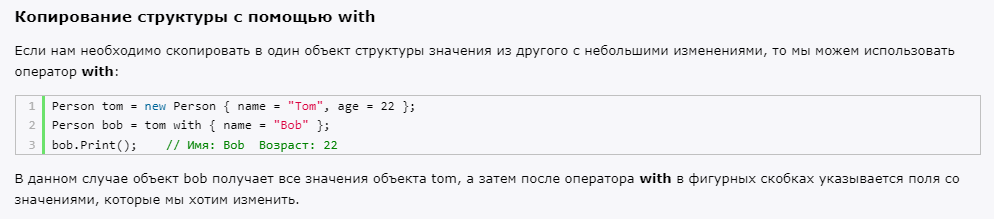








*Главное отличие* структур и классов: **структуры, указываемые в списке параметров метода, передаются по значению (то есть копируются), объекты классов — по ссылке.**



1. Что может и чего не может быть в структуре?

*Ответ:*

Как и классы, структуры могут хранить состояние в виде **полей (переменных) и** определять **поведение в виде методов.**

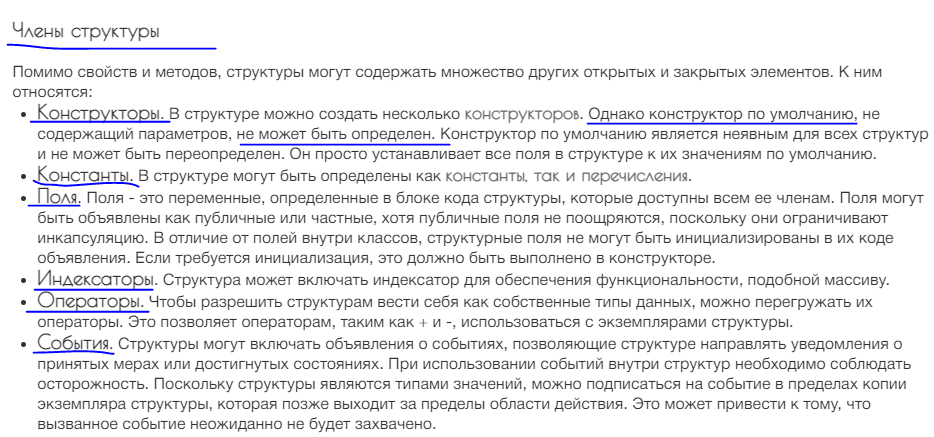
Структуры **могут иметь конструкторы. Однако, если в структуре определяется конструктор, то в нем обязательно надо инициализировать все поля структуры.**

Как и у классов, у каждой структуры имеются свои члены: **методы, поля, индексаторы, свойства, операторные методы и события**. В структурах допускается также определять **конструкторы**, **но не деструкторы**. В то же время для структуры **нельзя определить конструктор, используемый по умолчанию** (т.е. конструктор без параметров). Дело в том, что конструктор, вызываемый по умолчанию, определяется для всех структур автоматически и не подлежит изменению. Такой конструктор инициализирует поля структуры значениями, задаваемыми по умолчанию.

**Тип структуры не может наследовать от другого типа класса или структуры и не может быть базовым для класса. Однако тип структуры может реализовывать**[**интерфейсы**](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/interface)**.**

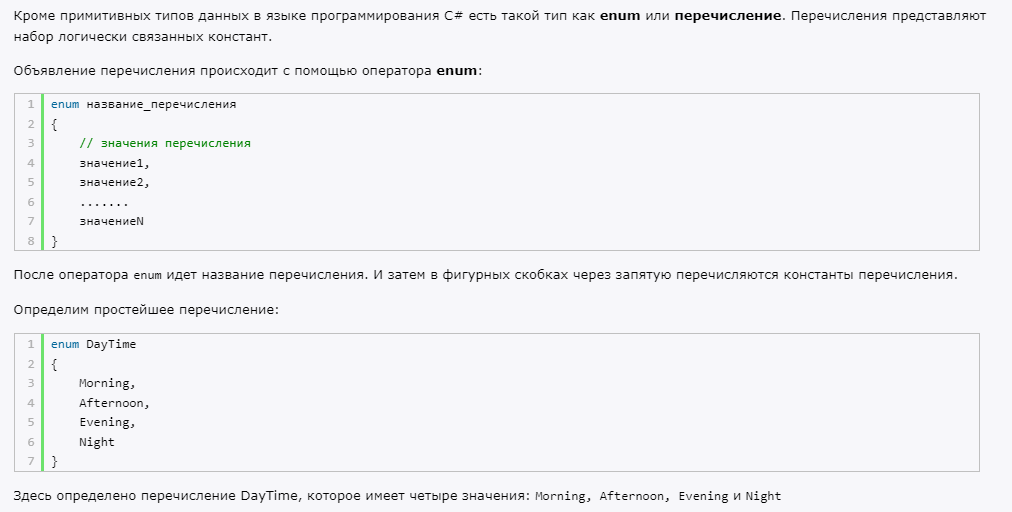
Структуры используют для обычной группировки логически связанных переменных и не более того.

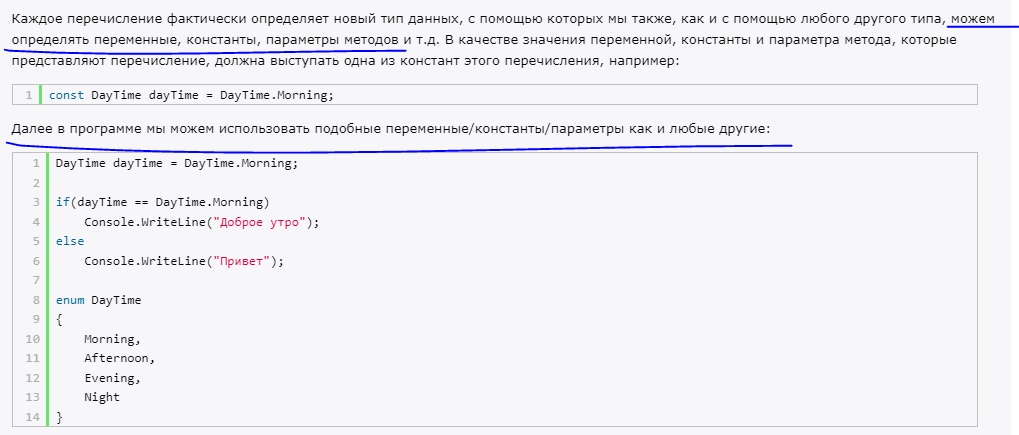
Исходя из тех соображений, что **структуры являются значимыми типами,** передача их в методы по умолчанию происходит по значению и изменения с переданными данными внутри метода не отобразятся на исходном объекте, в отличие от объектов классов.

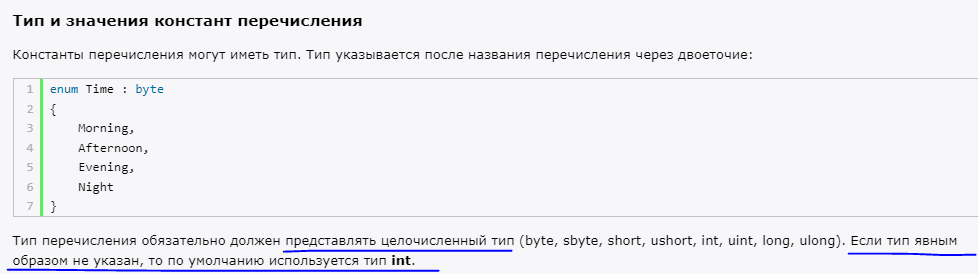


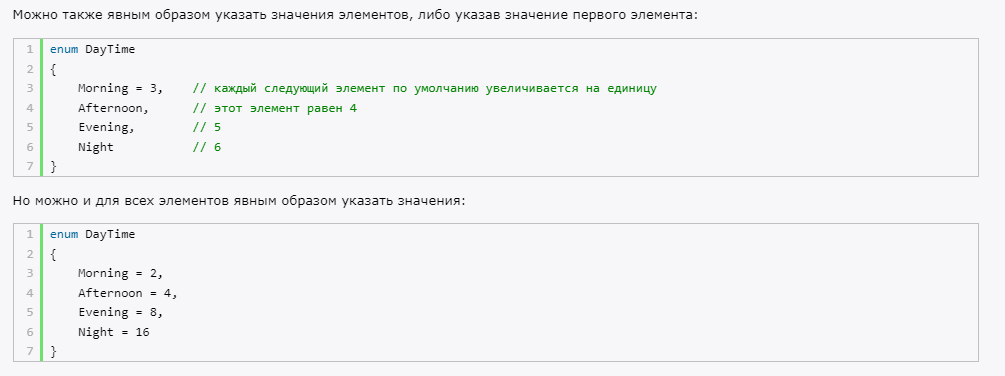
3. Что такое перечисление? Приведите пример определения и использования перечисления

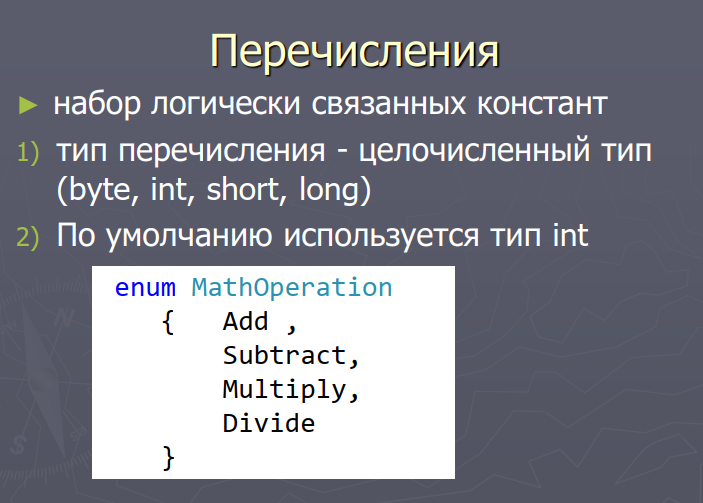
*Ответ:* **Перечисление (enumeration)** — это определяемый пользователем целочисленный тип.

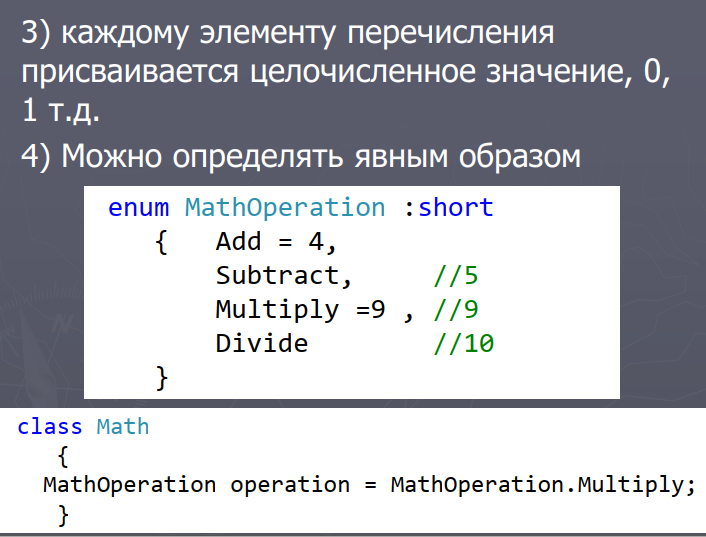




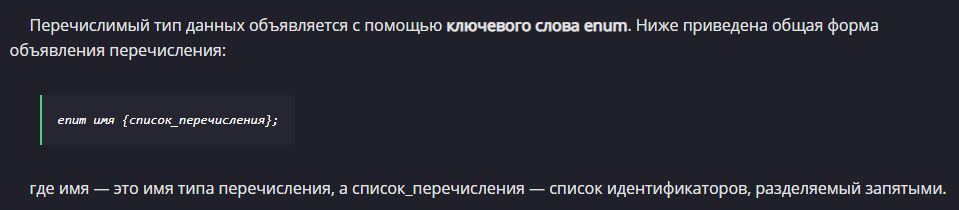








* **Перечисления делают код яснее**, позволяя **обращаться к целым значениям, называя их осмысленными именами** вместо малопонятных "магических" чисел.
* Как упоминалось, **перечисления облегчают сопровождение кода, гарантируя, что переменным будут присваиваться только легитимные, ожидаемые значения.**



**Перечисления в языке C# представляют собой набор (последовательность) констант, которые образуют тип данных**. Значение каждой константы задается именем.  В языке C# перечисления представлены классом. **Все перечисления наследуются от класса [System.Enum](https://www.bestprog.net/ru/2021/04/08/c-the-system-enum-class-enumeration-ru/" \t "_blank).**

**Перечисления C#** **— enum C# — это набор именованных констант, позволяющих сделать исходный код программы более удобочитаемым**. Оно позволяет задать имена для небольшого набора используемых целочисленных значений.

// Объявить тип Month

enum Month //ОБЪЯВЛЯЕМ НАШЕ ПЕРЕЧИСЛЕНИЕ

{

January, February, March, April, May, June,

July, August, September, October, November, December

}

// использование перечисления в программном коде

Month M; // Объявление переменной M типа "перечисление"

M = Month.January; // M = January

int m;

m = (int)M; // m = 0

M = Month.August; // M = August

m = (int)M; // m = 7

Инициализация перечисления:

// Объявить тип Month

enum Month

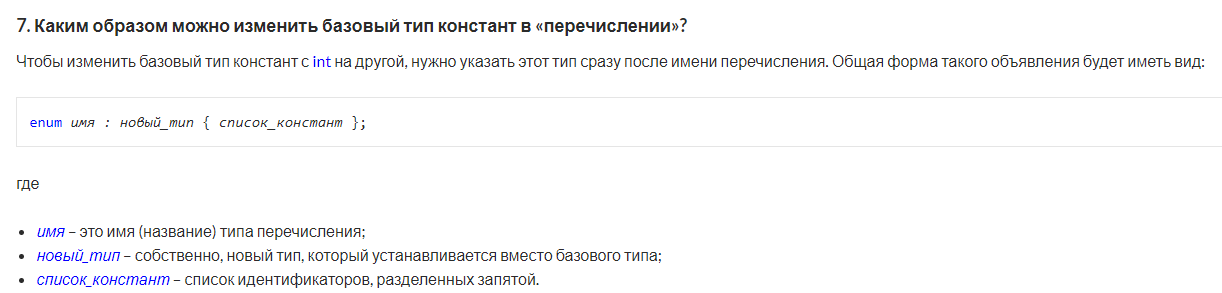
{

January = 31, February = 28, March = 31, April = 30, May, June = 30,

July, August = 31, September = 30, October, November = 30, December

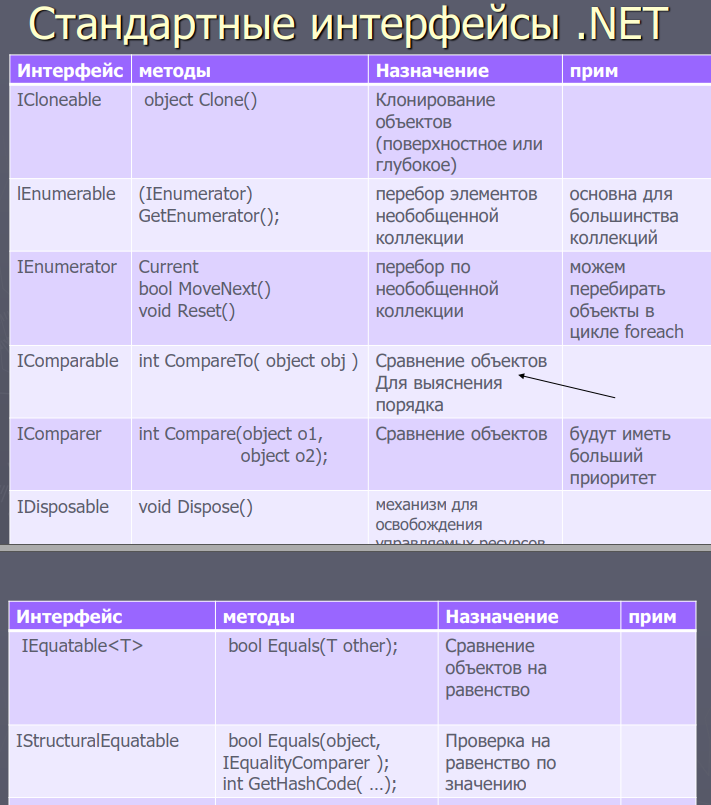
}

* **По умолчанию, базовым типом для констант в перечислении есть тип int.**



4. Перечислите и поясните стандартные интерфейсы .Net?

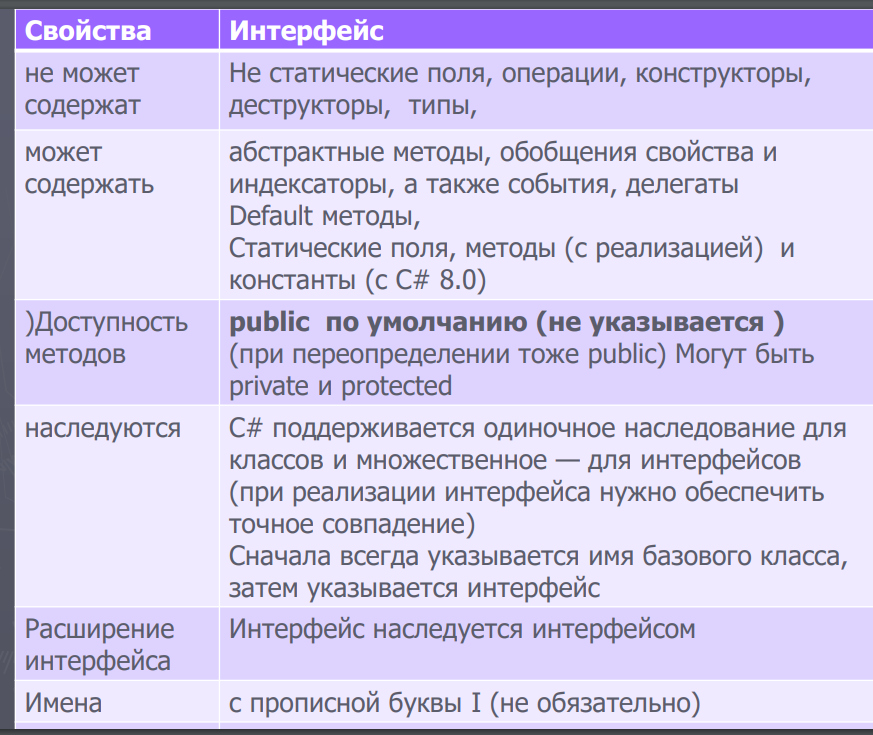
*Ответ:*

  
  
5. Как используется интерфейс IComparable?

*Ответ:*

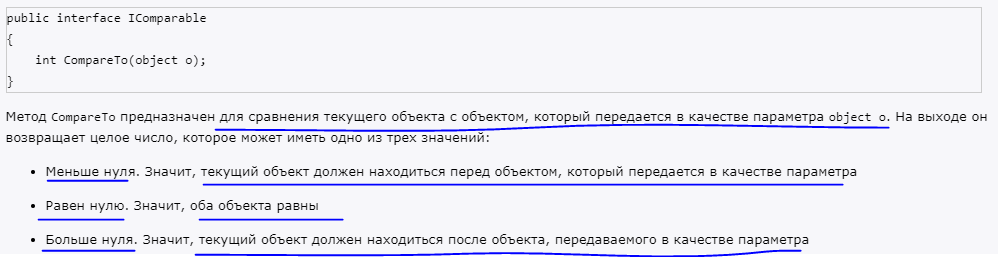
**Интерфейс представляет ссылочный тип, который может определять некоторый функционал - набор методов и свойств без реализации**. **Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы.**

Что вообще могут содержать интерфейсы?



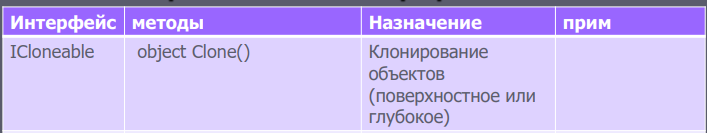
Теперь к сути вопроса:

Интерфейс IComparable используется **для сравнения объектов с целью выяснения их порядка**. **Имеет метод int ComapreTo(object obj).**



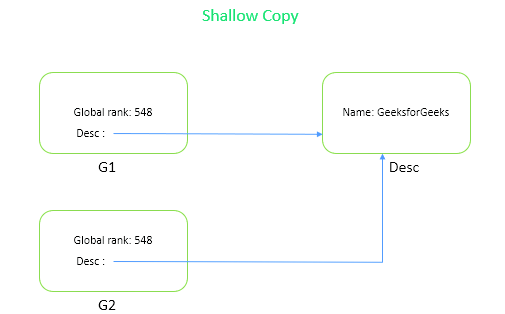
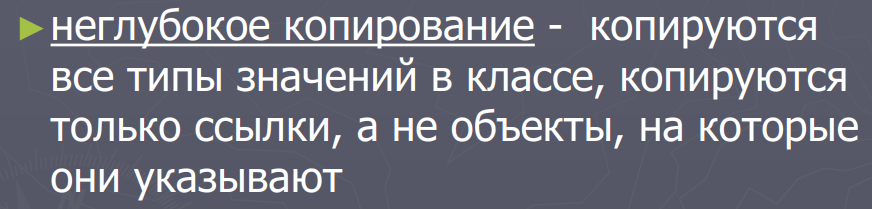
6. Как используется интерфейс ICloneable?

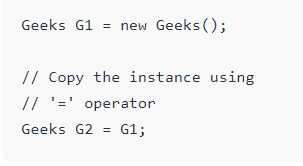
*Ответ:*

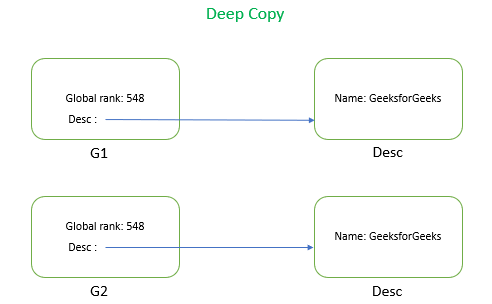


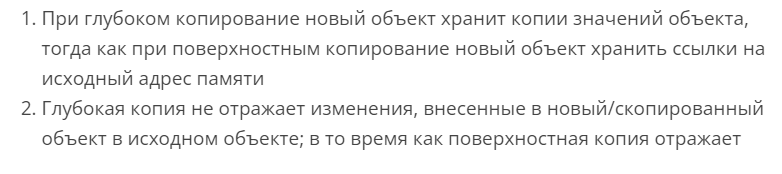
Что такое поверхностное и глубокое копирование?

**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. 



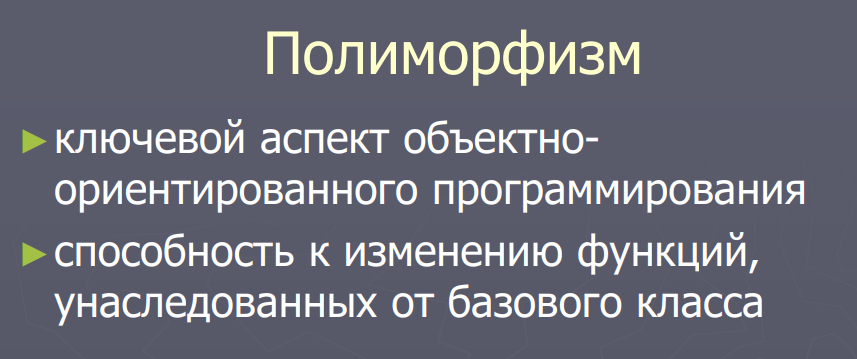
**Глубокая копия:** это процесс создания нового объекта с последующим копированием полей текущего объекта во вновь созданный объект для создания полной копии внутренних ссылочных типов. Если указанное поле является типом значения, то будет выполнено побитовое копирование поля. Если указанное поле является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

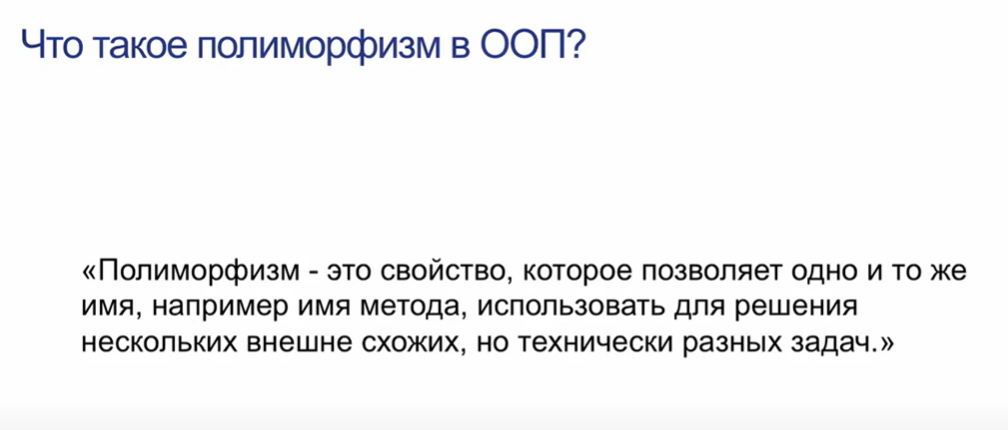


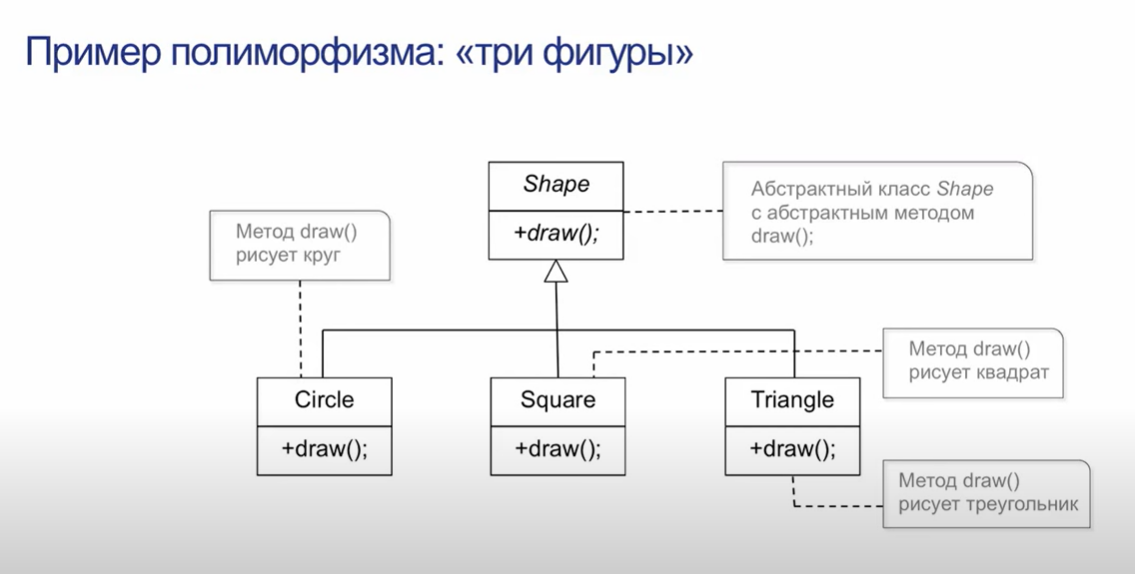
7. Что такое полиморфизм? Перечислите его формы. Приведите примеры.

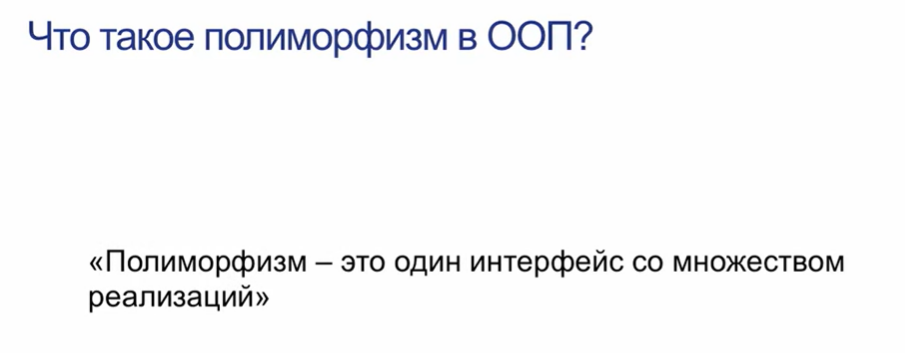
*Ответ:*

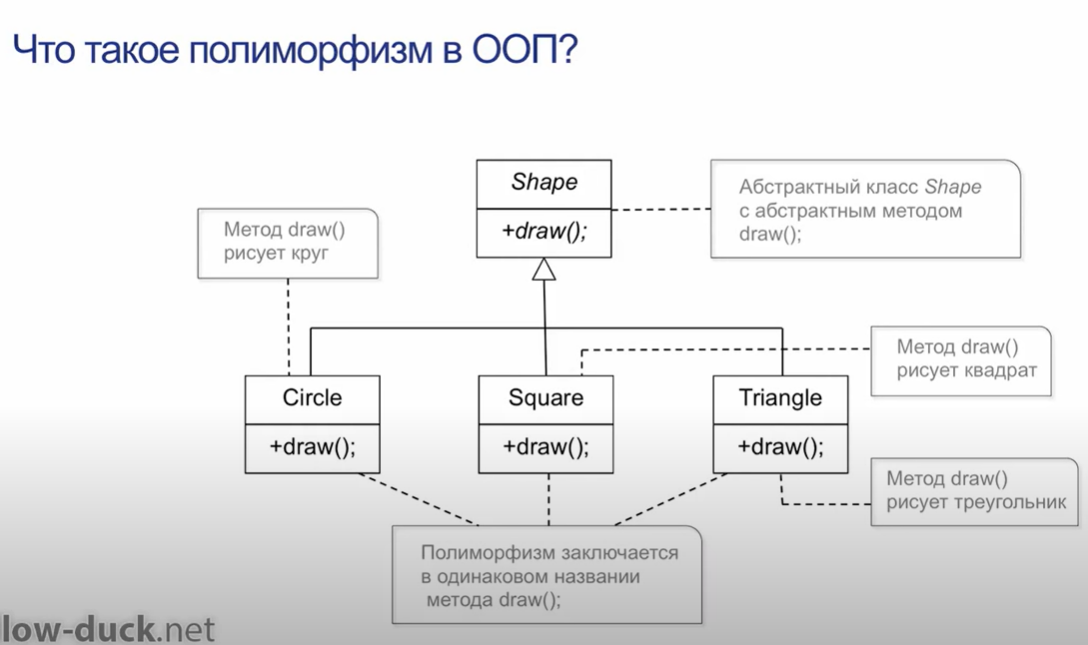
**Типичной фразой полиморфизм можно объяснить фразой "Один интерфейс, множество реализаций". В C# полиморфизм достигается за счет использования абстрактных и виртуальных методов.**

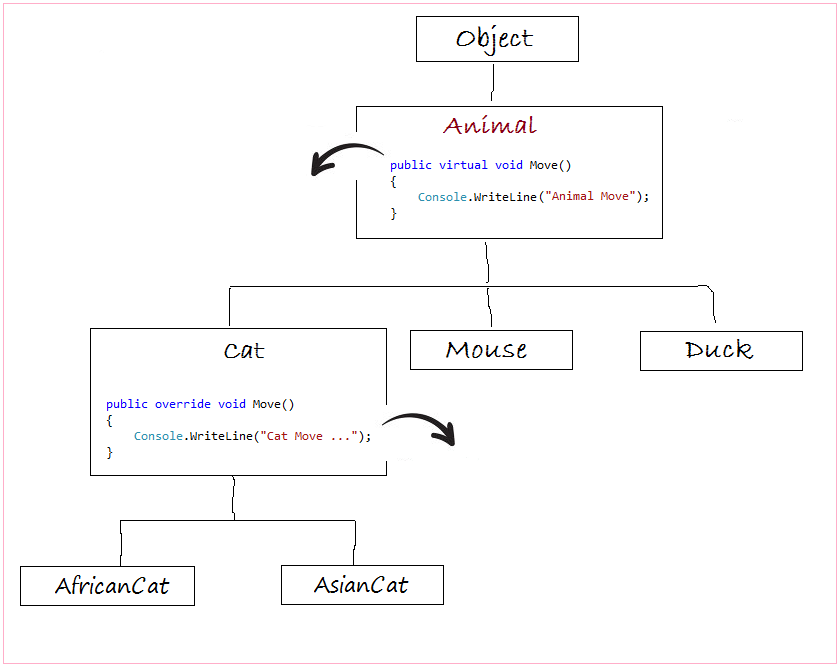


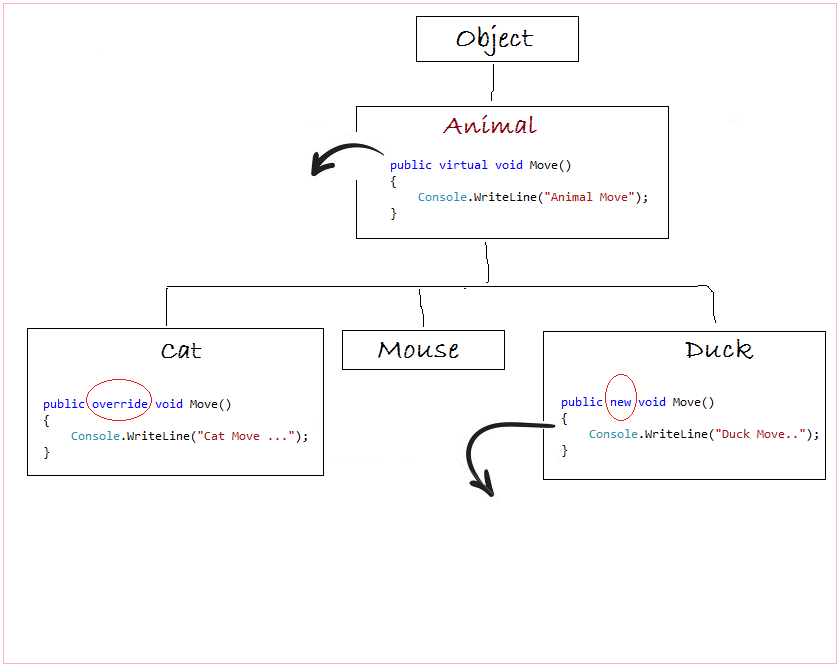






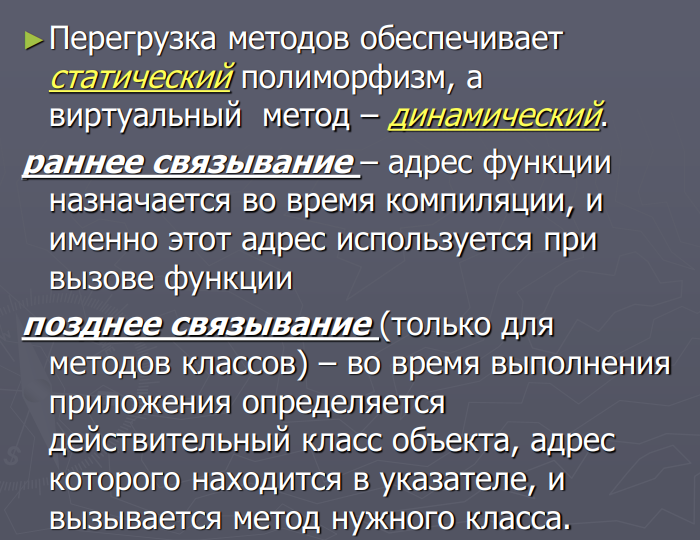






А если говорить простыми словами, **полиморфизм – это различная реализация однотипных действий**. Классическая фраза, которая коротко объясняет полиморфизм – «**Один интерфейс, множество реализаций».** Приведем примеры из жизни. «*В автомобилях есть рулевое колесо. Это колесо является интерфейсом между водителем и автомобилем, который позволяет поворачивать автомобиль. Механическая реализация руля у автомобилей может быть разная, но при этом результат получается одинаковым – колесо вправо – автомобиль вправо, и наоборот.»*

С полиморфизмом к нам прибавляются еще несколько понятий: **виртуальный/абстрактный метод, переопределение метода.**  
**Виртуальный метод – это метод, который МОЖЕТ быть переопределен в классе-наследнике**. Такой метод **может иметь стандартную реализацию** в базовом классе.  
  
**Абстрактный метод – это метод, который ДОЛЖЕН быть реализован в классе-наследнике.** При этом, абстрактный **метод не может иметь своей реализации** в базовом классе (тело пустое), в отличии от виртуального.  
  
**Переопределение метода – это изменение реализации метода, установленного как виртуальный** (в классе наследнике метод будет работать отлично от базового класса).

**Полиморфизм может быть статическим и динамическим.**   
  


СТАТИЧЕСКИЙ:

Статический полиморфизм:**решение о том, какой метод выполнить, принимается во время компиляции. Примером этого может быть перегрузка метода -** *это концепция, в которой мы используем одно и то же имя метода много раз в одном и том же классе, но с разными параметрами. В зависимости от параметров, которые мы передаем, решение принимается только во время компиляции.*

**Полиморфизм времени компиляции (статическое связывание / раннее связывание):** в статическом полиморфизме, если мы вызываем метод в нашем коде, **то определение того, какое определение этого метода должно быть вызвано на самом деле, разрешается только во время компиляции**.

ДИНАМИЧЕСКИЙ:

Динамический полиморфизм:**решение о выборе метода для выполнения устанавливается во время выполнения. Примером этого может быть переопределение метода.**

*Полиморфизм времени выполнения, также известный как* ***переопределение метода****. В этом механизме, с помощью которого вызов переопределенной функции разрешается во время выполнения (не во время компиляции), если базовый класс содержит переопределенный метод. Переопределение метода означает наличие двух или более методов с одинаковым именем, одинаковой сигнатурой, но с разной реализацией.*

8. Зачем в классе определяют виртуальные методы?

*Ответ:*

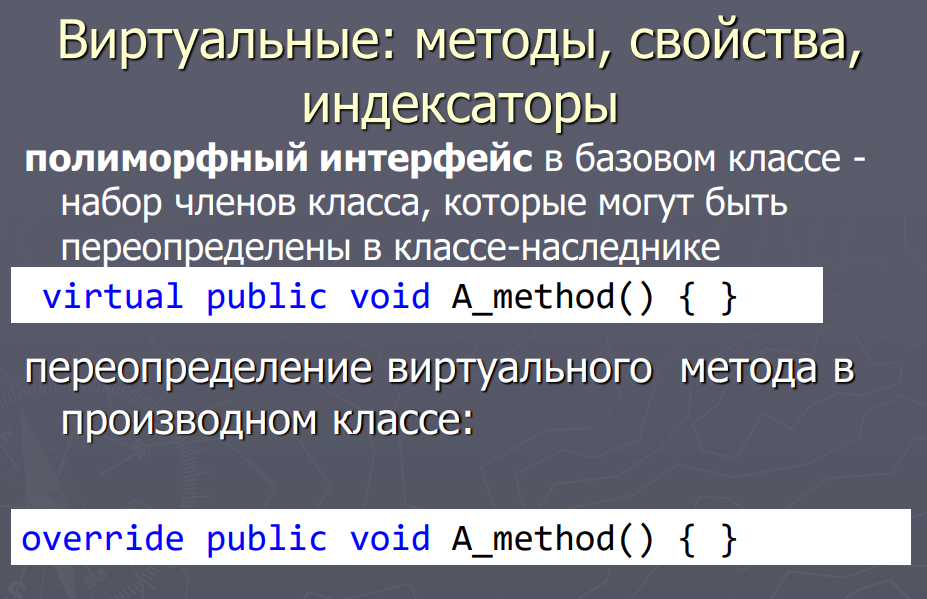
**При наследовании нередко возникает необходимость изменить в классе-наследнике функционал метода, который был унаследован от базового класса. В этом случае класс-наследник может переопределять методы и свойства базового класса.**

**Те методы и свойства, которые мы хотим сделать доступными для переопределения**, в базовом классе помечается модификатором **virtual**. Такие методы и свойства называют виртуальными.

**А чтобы переопределить метод в классе-наследнике, этот метод определяется с модификатором override**. **Переопределенный метод в классе-наследнике должен иметь тот же набор параметров, что и виртуальный метод в базовом классе.**

**Виртуальным называется такой метод, который объявляется как virtual в базовом классе**. Виртуальный метод отличается тем, что **он может быть переопределен в одном или нескольких производных классах.** ***Следовательно, у каждого производного класса может быть свой вариант виртуального метода.***

Кроме того, виртуальный метод не может быть объявлен как static или abstract.



**Главный принцип полиморфизма – один интерфейс, много реализаций (методов).**

**Виртуальные методы реализуют ДИНАМИЧЕСКИЙ ПОЛИМОРФИЗМ.**

**Сам процесс повторного определения виртуального метода в производном классе называется переопределением метода.**

9. Как сделать запрет переопределения методов?

*Ответ:*

**Запрет переопределения методов совершается с помощью ключевого слова seald.** Такой метод будет запечатанным.

В языке C# введено ключевое слово **sealed**, которое используется в двух случаях:

* **когда нужно запретить наследование от некоторого класса**. В многих иерархиях, классы, которые размещены на нижних уровнях могут быть обозначены как sealed;
* **когда нужно запретить переопределение некоторого метода в иерархии классов**. Эта ситуация возможна, если в иерархии классов методы унаследованных классов переопределяют виртуальные методы базовых классов.

