1. Назовите принципы ООП. Поясните каждый из них.

►Наследование (Inheritance); процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта.

►Инкапсуляция (Encapsulation); механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования

►Полиморфизм (Polymorphism) — это свойство одних и тех же объектов и методов принимать разные формы. возможность объектов с одинаковой спецификацией иметь различную реализацию. Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода.

►Абстракция данных; абстракция - это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

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

По умолчанию **все классы наследуются** от базового **класса** Object, даже если мы явным образом не устанавливаем **наследование**.

3. Охарактеризуйте открытые методы System.Object.

***bool Equals()*** Сравнивает две ссылки на объекты в период выполнения, чтобы определить, указывают ли они в точности один и тот же объект. Если две переменные ссылаются на один и тот же объект, возвращается true. В случае размерных типов (см. о них следующий раздел) этот метод возвращает true, если типы переменных идентичны и их значения равны.

***int GetHashCode*** Возвращает заданный для объекта хзш-код. Хэш-функции используются в реализации класса, когда хэш-код объекта нужно поместить в хэш-таблицу для повышения производительности.

***Type GetType()*** Используется с методами отражения для получения информации о типе данного объекта.

***string ToString*** Используется по умолчанию для получения имени объекта. Его можно переопределить в производных классах, чтобы они возвращали понятное пользователю текстовое представление объекта.

4. Охарактеризуйте закрытые методы System.Object.

1.Метод Описание

2.***void Finalize()*** Вызывается в период выполнения для освобождение ресурсов перед сбором мусора. Этот метод можно вызывать, а можно и не делать этого. Поэтому не помещайте в него подлежащий исполнению код. Это правило выливается в нечто под названием детерминированное завершение (deterministic finalization)

***3.Object MemberwiseClone*** Представляет ограниченную копию (shallow copy) объекта. Под этим я понимаю копию объекта, содержащую ссылки на другие объекты, но не копии этих объектов.

5. Приведите пример определения класса.

***class имя\_класса*** {  
 уровень\_доступа тип переменная1;  
 уровень\_доступа тип переменная2;  
 уровень\_доступа тип\_возвращаемого\_значени метод1(параметры) {  
//тело метода }   
уровень\_доступа тип\_возвращаемого\_значени метод2(параметры) {  
//тело метода} }

6. Какие ключевые слова можно использовать при определении класса?

Ключевому слову class предшествует уровень доступа. Поскольку в этом случае используется открытый класс (public), любой пользователь может создавать объекты из этого класса. За именем класса следует ключевое слово class. Оставшаяся часть определения — это тело класса, в котором задаются данные и поведение. Поля, свойства, методы и события в классе собирательно называются членами класса.

class

7. В чем отличие между объектом и классом?

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

Объект – это физическая реализация класса(шаблона).

Определение класса или структуры подобно чертежу, на котором указаны действия, выполняемые типом. В сущности, объект является блоком памяти, выделенной и настроенной в соответствии с чертежом. Программа может создать множество объектов одного класса. Объекты также называют экземплярами. Они могут храниться либо в именованной переменной, либо в массиве или коллекции. Клиентский код — это код, использующий эти переменные для вызова методов и доступа к открытым свойствам объекта. В объектно-ориентированном языке, таком как C#, стандартная программа состоит из нескольких динамически взаимодействующих объектов.

Класс — это конструкция, которая позволяет создавать собственные настраиваемые типы путем группирования переменных других типов, методов и событий. Класс похож на проект. Он определяет данные и поведение типа. Если класс не объявлен как статический, клиентский код может использовать его, создавая объекты или экземпляры, назначенные переменной. Переменная остается в памяти до тех пор, пока все ссылки на нее не выйдут из области. В этот момент среда CLR помечает ее пригодной для сборки мусора. Если класс объявляется как статический, то в памяти существует только одна копия, и клиентский код может получать к ней доступ только через сам класс, а не переменную экземпляра.

8. Что такое конструктор? Когда вызывается конструктор?

**Конструкторы** — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа. Конструктор инициализирует объект при его создании. У конструктора такое же имя, как и у его класса, а с точки зрения синтаксиса он подобен методу. Но у конструкторов нет возвращаемого типа, указываемого явно. Ниже приведена общая форма конструктора:

**доступ имя\_класса(список\_параметров) {**

**// тело конструктора**

**}**

Конструктор также может принимать один или несколько параметров. В конструктор параметры вводятся таким же образом, как и в метод. Для этого достаточно объявить их в скобках после имени конструктора.

9. Перечислите свойства конструктора?

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:

10. Что такое деструктор (destructor) ?

вызываться непосредственно перед окончательным уничтожением объекта системой "сборки мусора", чтобы гарантировать четкое окончание срока действия объекта.

~имя\_класса () { // код деструктора }

**Деструктор** — это функция-член, которая вызывается автоматически, когда объект выходит из области действия или явно уничтожается вызовом метода delete .

11. Что такое this?

обеспечивает доступ к текущему экземпляру класса.   
в любой нестатический метод автоматически передается скрытый параметр this

12. Что будет выведено в результате выполнения

class A

{

private int \_num;

public A(int num) { Num = num; }

public int Num { get { return \_num; } set { \_num = value; } }

}

static void Main(string[] args)

{

A a = new A(5);

A b = a;

Console.WriteLine(a.Num + " " +b.Num);

a.Num = 7;

Console.WriteLine(a.Num + " " + b.Num);

}

**Ответ:**

5 5

7 7

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

Public, Private, protected, internal

14. Опишите модификатор protected internal.

совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов

Все типы и члены имеют уровень доступности, определяющий возможность их использования из другого кода в вашей или в других сборках. Следующие модификаторы доступа позволяют указать доступность типа или члена при объявлении:

public

Доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.

private

Доступ к типу или члену возможен только из кода в том же классе или структуре.

protected

Доступ к типу или члену возможен только из кода в том же классе или структуре либо в классе, производном от этого класса.

internal

Доступ к типу или члену возможен из любого кода в той же сборке, но не из другой сборки.

protected internal

Доступ к типу или члену возможен из любого кода в той сборке, где он был объявлен, или из производного класса в другой сборке. Доступ из другой сборки должен осуществляться в объявлении класса, производного от класса, в котором объявлен защищенный внутренний элемент, и через экземпляр типа производного класса.

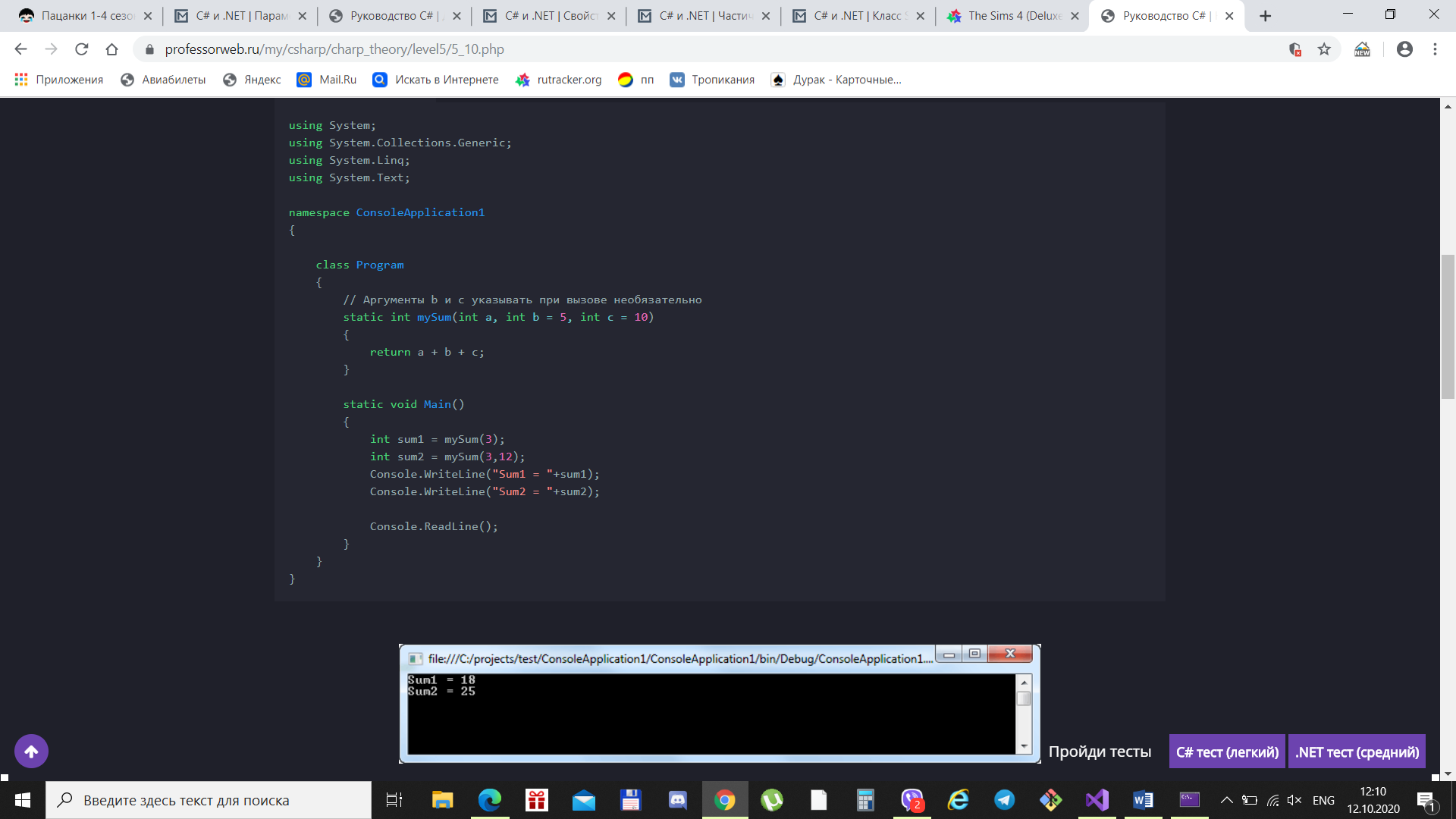
15. Зачем и как используются ref и out параметры функции?

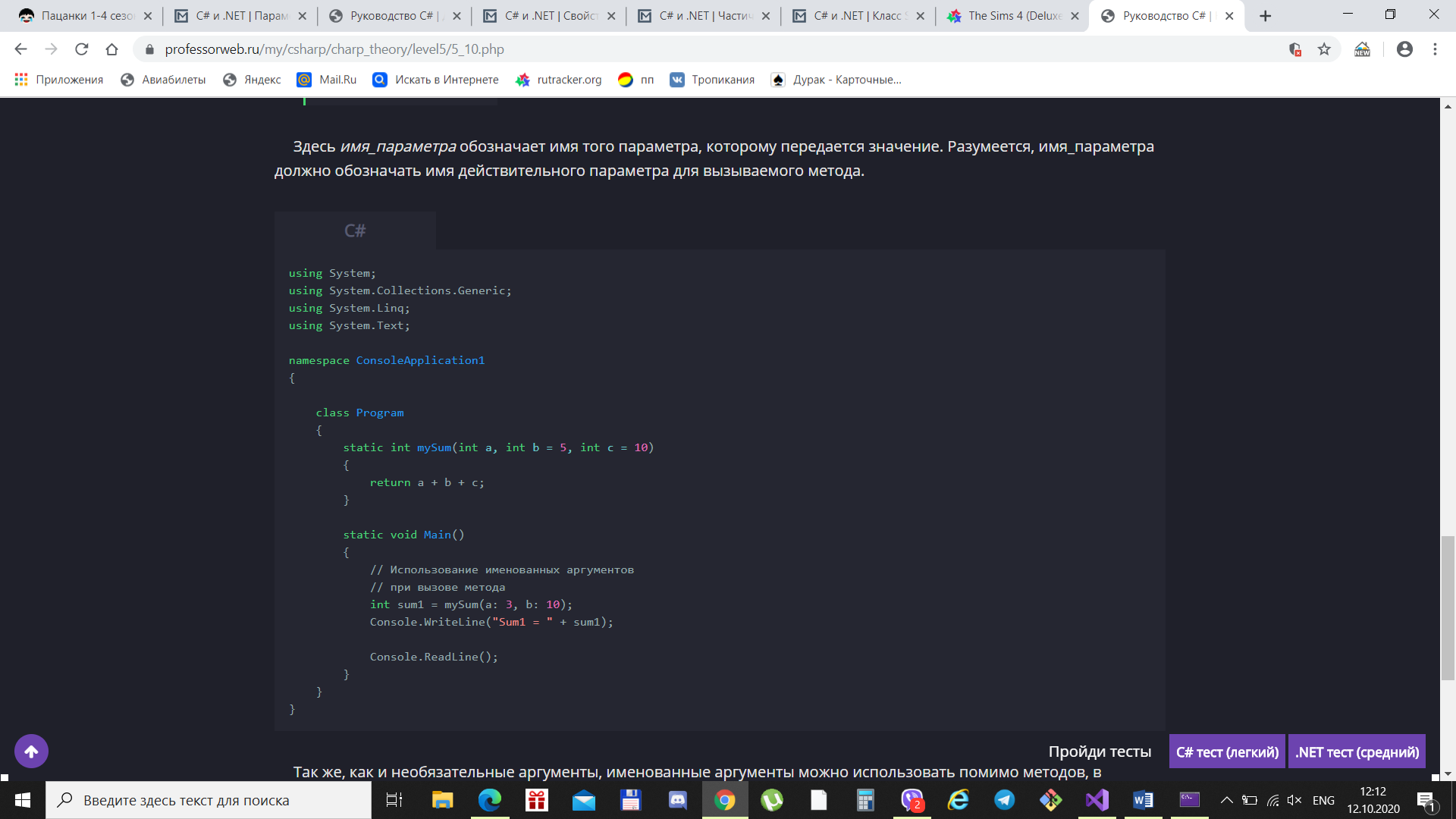
При передаче параметров по ссылке перед параметрами используется модификатор **ref**. Но параметры могут быть также выходными. Чтобы сделать параметр выходным, перед ним ставится модификатор out

Разница в том, что out — это выходной параметр, а ref — входно-выходной.

Для ref-параметра вы должны передать его инициализированным, и можете пользоваться его исходным значением. А для out-параметра вы не обязаны инициализировать его перед вызовом функции, не можете использовать его значение в функции до присваивания, и обязаны инициализировать его в функции.

16. Приведите пример необязательных и именованных параметров метода.





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

Модификатор static используется для объявления статического члена, принадлежащего собственно типу, а не конкретному объекту. Модификатор static можно использовать с классами, полями, методами, свойствами, операторами, событиями и конструкторами, но нельзя — с индексаторами, методами завершения или типами, отличными от классов.

Пример

Следующий класс объявляется как static и содержит только методы static:

C#

Копировать

static class CompanyEmployee

{

public static void DoSomething() { /\*...\*/ }

public static void DoSomethingElse() { /\*...\*/ }

}

Объявление константы или типа неявно является статическим членом.

Для объявления константного поля или константной локальной используется ключевое слово const. Константные поля и локальные не являются переменными и не могут быть изменены. Константы могут быть числами, логическими значениями, строками или нулевыми ссылками. Не создавайте константу для предоставления сведений, которые могут измениться в любое время. Например, не используйте константное поле для хранения цены услуги, номера версии продукта или торгового названия компании. Эти значения могут со временем измениться, а поскольку константы распространяются компиляторами, для отражения изменений потребуется повторная компиляция остальных кодов, скомпилированных с использованием ваших библиотек. См. также описание ключевого слова readonly. Пример:

C#

Копировать

const int x = 0;

public const double gravitationalConstant = 6.673e-11;

private const string productName = "Visual C#";

Ключевое слово readonly — это модификатор, который можно использовать для полей. Если объявление поля содержит модификатор readonly, присвоение значений таким полям может происходить только как часть объявления или в конструкторе в том же классе.

Пример

В этом примере значение поля year нельзя изменить в методе ChangeYear, несмотря на то, что в конструкторе класса ему присваивается значение:

C#

class Age

{

readonly int \_year;

Age(int year)

{

\_year = year;

}

void ChangeYear()

{

//\_year = 1967; // Compile error if uncommented.

}

}

18. Приведите пример определения свойств класса. Как свойства связаны с инкапуляцией?

**Инкапсуляция** — в информатике размещение в одном компоненте данных и методов, которые с ними работают.

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

value. Тип этого параметра определяется типом свойства.

20. Что такое автоматические свойства? **{get ; set}**  
**Автоматическое** **свойство** – это очень простое **свойство**, которое, в отличии от обычного **свойства**, уже определяет место в памяти (создает неявное поле), но при этом не позволяет создавать логику доступа. Структура объявления **Автоматического** **свойства**: [модификатор доступа] [тип] [имя\_**свойства**] { get; set; }

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

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

Программисты хорошо знакомы с процессом доступа к индивидуальным элементам, содержащимся в стандартных массивах, через операцию индекса ([]). В C# имеется возможность проектировать специальные классы и структуры, которые могут быть индексированы подобно стандартному массиву, посредством определения индексатора. Это конкретное языковое средство наиболее полезно при создании специальных типов коллекций (обобщенных и необобщенных). Индексаторы могут быть одно- или многомерными.

Позволяют индексировать объекты таким же способом, как массив или коллекцию;

Ниже приведена общая форма одномерного индексатора:

тип\_элемента this[int индекс] {

// Аксессор для получения данных,

get {

// Возврат значения, которое определяет индекс.

}

// Аксессор для установки данных,

set {

// Установка значения, которое определяет индекс.

}}

где тип\_элемента обозначает конкретный тип элемента индексатора. Следовательно, у каждого элемента, доступного с помощью индексатора, должен быть определенный тип\_элемента. Этот тип соответствует типу элемента массива. Параметр индекс получает конкретный индекс элемента, к которому осуществляется доступ. Формально этот параметр совсем не обязательно должен иметь тип int, но поскольку индексаторы, как правило, применяются для индексирования массивов, то чаще всего используется целочисленный тип данного параметра.

22. Что такое перегруженный метод?

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**

В C# допускается совместное использование одного и того же имени двумя или более методами одного и того же класса, при условии, что их параметры объявляются по-разному. В этом случае говорят, что методы перегружаются, а сам процесс называется перегрузкой методов. Перегрузка методов относится к одному из способов реализации полиморфизма в C#.

В общем, для перегрузки метода достаточно объявить разные его варианты, а об остальном позаботится компилятор. Но при этом необходимо соблюсти следующее важное условие: тип или число параметров у каждого метода должны быть разными.

Совершенно недостаточно, чтобы два метода отличались только типами возвращаемых значений. Они должны также отличаться типами или числом своих параметров. (Во всяком случае, типы возвращаемых значений дают недостаточно сведений компилятору C#, чтобы решить, какой именно метод следует использовать.) Разумеется, перегружаемые методы могут отличаться и типами возвращаемых значений. Когда вызывается перегружаемый метод, то выполняется тот его вариант, параметры которого соответствуют (по типу и числу) передаваемым аргументам.

23. Что такое partial класс и какие его преимущества?

Частичные классы (partial class) представляют возможность разделить функционал одного класса на несколько файлов. Например, сейчас у нас код класса Book весь находится в одном файле Book.cs. Но мы можем разделить весь код на несколько разных файлов. В этом случае нам надо будет поставить перед определением класса ключевое слово partial. Допустим в одном файле будет:

partial class Book

{

public string name;

public string author;

public int year;

}

24. Что такое анонимный тип в C#?

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.

При этом во время компиляции компилятор сам будет создавать для него имя типа и использовать это имя при обращении к объекту. Нередко анонимные типы имеют имя наподобие "<>f\_\_AnonymousType0'2".

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

Если в программе используются несколько объектов анонимных типов с одинаковым набором свойств, то для них компилятор создаст одно определение анонимного типа:

25. Для чего делают статические классы?

Статический класс в основном такой же, как и нестатический класс, но имеется одно отличие: нельзя создавать экземпляры статического класса. Другими словами, нельзя использовать ключевое слово new для создания переменной типа класса. Поскольку нет переменной экземпляра, доступ к членам статического класса осуществляется с использованием самого имени класса. Например, если имеется статический класс, называемый UtilityClass, имеющий открытый метод с именем MethodA, вызов метода выполняется, как показано в следующем примере:

C#

Статические классы объявляются с модификатором static и могут содержать только статические поля, свойства и методы.

Назначение:

1) при создании метода расширения

2) для хранения совокупности связанных друг с другом статических методов

26. В чем отличие статического поля от экземплярного?

Можно понимать **статические** члены, как принадлежащие классам, а **экземплярные** члены как принадлежащие объектам

Статический метод - вызывается на уровне типа (вызов: Имя\_типа точка Имя\_метода)

Экземплярный метод - вызывается на уровне объявленного экземпляра типа (вызов: Имя\_переменной точка Имя\_метода).

27. Поясните работу статических конструкторов.

Статический конструктор используется для инициализации любых статических данных или для выполнения определенного действия, которое требуется выполнить только один раз. Он вызывается автоматически перед созданием первого экземпляра или ссылкой на какие-либо статические члены.

C#

class SimpleClass

{

// Static variable that must be initialized at run time.

static readonly long baseline;

// Static constructor is called at most one time, before any

// instance constructor is invoked or member is accessed.

static SimpleClass()

{

baseline = DateTime.Now.Ticks;

}

}

Статические конструкторы обладают следующими свойствами.

Статический конструктор не принимает модификаторы доступа и не имеет параметров.

Статический конструктор вызывается автоматически для инициализации класса перед созданием первого экземпляра типа или ссылкой на какие-либо статические члены.

Статический конструктор нельзя вызывать напрямую.

Пользователь не управляет временем, в течение которого статический конструктор выполняется в программе.

Типичным использованием статических конструкторов является случай, когда класс использует файл журнала и конструктор применяется для добавления записей в этот файл.

Статические конструкторы также полезны при создании классов-оболочек для неуправляемого кода, когда конструктор может вызвать метод LoadLibrary.

Если статический конструктор инициирует исключение, среда выполнения не вызывает его во второй раз, и тип остается неинициализированным на время существования домена приложения, в котором выполняется программа.

Пример

В этом примере класс Bus имеет статический конструктор. При создании первого экземпляра класса Bus (bus1) для инициализации класса вызывается статический конструктор. В выходных данных этого примера можно увидеть, что статический конструктор выполняется только один раз, несмотря на то, что создается два экземпляра класса Bus. Кроме того, этот конструктор вызывается до выполнения конструктора экземпляра.

C#

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

При поверхностном копировании копируются значения полей класса, включая значения любых указателей или ссылок. При этом скопированные значения этих указателей и ссылок указывают на одни и те же объекты, что и в оригинальном объекте, что зачастую ведет к ошибкам. Отсюда и название такого метода копирования: мы копируем только указатели/ссылки, вместо того, чтобы делать копии этих внутренних объектов и ссылаться на них, собственно не углубляемся во внутреннюю структуру объекта. При глубоком копировании мы копируем значения полей не только на первом "уровне", но и заходим глубже, копируя все значения.

29. В чем разница между равенством и тождеством объектов?

Понятие и знак "равно" используют:

а) для записи верных числовых равенств, например, 3+2=5.

б) для записи уравнений, т. е. соотношений, содержащих переменные

величины, причем такое соотношение может быть верным при

некоторых значениях переменных, называемых решениями (или

корнями) уравнения. Например, 3х=9 верно при х=3.

Понятие "тождество" (или тождественное равенство, обозначают

тремя черточками) означает равенство, содержащее переменные

величины, причем это равенство должно быть верным при всех

(допустимых) значениях переменных, например, 3(х+у) =3х+3у.

30. Что такое частичные классы и частичные методы?

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.

Есть несколько ситуаций, когда удачно разбить определение класса на несколько файлов:

1. При работе над большим проектом, разбиение класса на несколько файлов позволяет нескольким программистам работать над ним одновременно.

2. При работе с автоматически генерируемыми исходниками, код может быть добавлен в класс без необходимости пересоздавать исходник. Visual Studio использует этот подход при создании компонентов Windows Forms, веб-сервисов и т.д. Возможно создать код, который использует эти классы без необходимости редактировать файлы, которые создает Visual Studio.

Для такого разбития класса используется модификатор partial.

астичные методы обычно используются кодогенераторами. Они объявляются в автоматически сгенерированной части кода и оттуда же вызваются, а в части кода, которая пишется вручную, их можно реализовать. Если метод не реализован, при компиляции его вызовы автоматически удаляются из кода. Соответственно, сигнатура этих методов должна быть такой, чтобы их удаление никак не повлияло на собираемость кода. Отсюда описанные ограничения.

31. Что будет выведено на консоль результате выполнения следующего кода:

int

32. Что будет выведено на консоль результате выполнения следующего кода:

a=2; b=1;

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

2

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

Может 1, может быть и 2

35. Почему не удается создать объект класса A?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Серьезность | Код | Описание | Проект | Файл | Строка | Состояние подавления | Состояние подавления |
| Ошибка | CS0122 | 'A.A(int)" недоступен из-за его уровня защиты. | ConsoleApp1 | C:\Users\thesi\Desktop\ConsoleApp1\ConsoleApp1\Program.cs | 16 |  | Активный |

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

A staticA

37. Какая строка приведенного далее класса вызовет ошибку компиляции?

3