

1. Назовите принципы ООП. Поясните каждый из них.

- ▶ Наследование (Inheritance);
- ▶ Инкапсуляция (Encapsulation);
- ▶ Полиморфизм (Polymorphism).
- ▶ Абстракция данных

Наследование:

- ▶ процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта.

Инкапсуляция:

- ▶ механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования

Полиморфизм:

- ▶ -способность вызывать метод потомка через экземпляр предка

Абстракция данных:

Абстракция подразумевает разделение и независимое рассмотрение **интерфейса** и **реализации**

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

3. Охарактеризуйте открытые методы System.Object.

класс System.Object определяет шесть открытых методов:

`public virtual bool Equals(object object)`

Этот метод определяет равенство вызывающего и передаваемого экземпляров.

`public static bool Equals(object ob1, object ob2)`

Этот метод определяет равенство объектов ob1 и ob2, передаваемых в качестве

параметров. Метод Equals() перегружен, т.к. для него определено два разных контекста.

```
public virtual int GetHashCode()
```

Этот метод возвращает хэш-код, соответствующий вызываемому объекту. Применяется в алгоритмах, использующих хэши для доступа к экземплярам.

```
public Type GetType()
```

Этот метод возвращает тип объекта.

```
public static bool ReferenceEquals(object ob1, object ob2)
```

Этот метод определяет равенство ссылок ob1 и ob2, т.е. ссылаются ли они на один экземпляр.

```
public virtual string ToString()
```

Этот метод возвращает строку с описанием объекта.

4. Охарактеризуйте закрытые методы System.Object.

И два закрытых:

```
protected void Finalize()
```

Вызывается перед сборкой мусора. В принципе, можно не использовать. Не следует помещать в выполняемый код. Обычно вызывается деструктором.

```
protected object MemberwiseClone()
```

метод возвращает значение типа object. Этот метод скорее создает копию объекта, содержащую ссылки на другие объекты.

5. Приведите пример определения класса.

```
Public class Student
```

```
{  
}
```

6. Какие ключевые слова можно использовать при определении класса?

Ключевому слову `class` предшествует уровень доступа(public, private, protected, internal).

За именем класса следует ключевое слово `class`. Оставшаяся часть определения — это тело класса, в котором задаются данные и поведение.

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).

8. Что такое конструктор? Когда вызывается конструктор?

Конструкторы — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа.

Каждый раз, когда создается [класс](#) или [структура](#), вызывается конструктор.

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) ?

Деструктор — это метод для деинициализации объекта.

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

```
~имя_класса () { // код деструктора }
```

11. Что такое `this`?

Ключевое слово, которое обеспечивает доступ к текущему экземпляру класса

12. Что будет выведено в результате выполнения

5 5

7 7

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

- ▶ **public** - доступ не ограничен — все методы 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

Модификаторы определяют, на какие члены можно ссылаться из кода

14. Опишите модификатор `protected internal`.

[`protected internal`](#): доступ ограничен текущей сборкой или типами, которые являются производными от содержащего класса.

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

- ▶ ***ref** заставляет C# организовать вместо вызова по значению вызов по ссылке*

► Модификатор *out* подобен модификатору *ref* за одним исключением:

его можно использовать для передачи значения из метода


out-параметр "поступает" в метод без начального значения, но метод (до своего завершения) **обязательно** должен присвоить этому параметру значение

16. Приведите пример необязательных и именованных параметров метода.

Необязательные аргументы

- позволяет определить используемое по умолчанию значение для параметра метода
- можно применять в конструкторах, индексаторах

```
static void RedrawButton(int color ,  
                        int type = 2 ,  
                        int size = 4)  
{ }
```



Именованные аргументы

значение аргумента присваивается параметру по его позиции в списке аргументов

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

```
static void RedrawButton(int color ,
                        int type = 2 ,
                        int size = 4)
{ }
static void Main(string[] args)
{
    RedrawButton(243, size:45);
```

порядок следования аргументов не имеет значения

17. Приведите пример полей класса – статические, константные, только для чтения. Нестатический класс может содержать статические методы, поля, свойства или события. Статический член вызывается для класса даже в том случае, если не создан экземпляр класса. Доступ к статическому члену всегда выполняется по имени класса, а не экземпляра. Существует только одна копия статического члена, независимо от того, сколько создано экземпляров класса.

Статические поля обычно используются для следующих двух целей: хранение счетчика числа созданных объектов или хранение значения, которое должно совместно использоваться всеми экземплярами.

Public static int count = 0;

Константы — это постоянные значения, которые известны во время компиляции и не изменяются во время выполнения программы. Константы должны объявляться с модификатором [const](#). Только встроенные типы C# (за исключением [System.Object](#)) можно объявлять как `const`

```
Const int = 100;
```

Модификатор [readonly](#) позволяет создать класс, структуру или массив, которые инициализируются один раз (например, в конструкторе), и впоследствии изменить их нельзя.

Поля для чтения

readonly - инициализация времени испол.

- 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.

```
class Point
{
    public int x;
    public readonly int y = 0; // можно так инициализировать
    public Point (int _y)
    {
        y = _y; //может быть инициализировано
    } //или изменено в конструкторе после компиляции
```

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

Свойства класса

- 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).
- Синтаксис свойства:

[атрибуты] [спецификаторы] тип имясвойства

```
{
    [get код_доступа]
    [set код_доступа]
}
```

не void

аксессоры


```
class StudentBSTU
```

```
{
```

```
    private string name;
```

Закрытое поле

```
    public string Name
```

Имя - произвольное и не обязательно должно совпадать.

```
{
```

Свойство

```
        get
```

Способ
получения
свойства

```
{
```

```
            return name;
```

```
}
```

```
        set
```

Способ
установки
свойства

```
{
```

```
            name = value;
```

«умные» поля, то есть полями с дополнительной логикой

```
}
```

представляет передаваемое значение

```
}
```

19. Назовите явное имя параметра, передаваемого в метод set свойства класса? Value. Тип этого параметра определяется типом свойства.

20. Что такое автоматические свойства?
Свойства, при которых

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

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

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

22. Что такое перегруженный метод?

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

23. Что такое partial класс и какие его преимущества?

возможность разделить функционал одного класса на несколько файлов.

Можно разделить определение [класса](#) или [структуры](#), [интерфейса](#) или метода между двумя или более исходными файлами. Каждый исходный файл содержит часть определения класса или метода, а во время компиляции приложения все части объединяются.

Частичные классы структуры, интерфейсы и методы

Назначение:

- ▶ Управление версиями
- ▶ Разделение файла или структуры на логические модули
- ▶ Использование шаблонов (авто генерируемый код)

Объединение всех частичных файлов класса во время компиляции;
CLR всегда работает с полными определениями типов.

24. Что такое анонимный тип в C#?

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

25. Для чего делают статические классы?

Назначение:

1) при создании *метода расширения*

2) для хранения совокупности связанных друг с другом статических методов

26. В чем отличие статического поля от экземплярного?

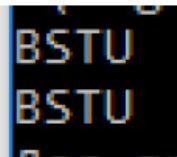
Статические поля:

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

► При использовании статических членов необязательно создавать экземпляр класса

```
StudentBSTU.getUo();
```

```
Console.WriteLine(StudentBSTU.UO);
```



BSTU
BSTU

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

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.

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

28. Какая разница между поверхностным (shallow) и глубоким (deep) копированием? Копирование бывает двух типов - глубокое и поверхностное. При глубоком создаётся абсолютно отдельная копия, при поверхностном создаётся копия класса, но она содержит ссылки на те же объекты что и оригинал

- **Поверхностная копия** создает новый составной объект, и затем (по мере возможности) вставляет в него ссылки на объекты, находящиеся в оригинале.
- **Глубокая копия** создает новый составной объект, и затем рекурсивно вставляет в него копии объектов, находящихся в оригинале.

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?

Тождество - сравниваются ссылки

Равенство – сравниваются **объекты**

30. Что такое частичные классы и частичные методы?

Ключевое слово `partial` позволяет определить *частичный класс*, а весь класс будет распределен по нескольким файлам исходного кода. Таким образом, одну часть класса можно сгенерировать автоматически, а другую — запрограммировать вручную.

К частичным классам предъявляется требование, чтобы они были полностью определены в сборке.

Ключевое слово `partial` можно также применять для создания методов, которые определены в одном месте, а реализованы в другом. Частичные методы можно рассматривать как определение метода абстрактного класса и реализация его в этом же методе.

Частичные методы определяются в контексте частичного класса. Частичный метод объявляется и обозначается префиксным идентификатором `partial` в одной части класса, а реализуется в другой.

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


```

class Program
{
    static void Main(string[] args)
    {
        var age = 15;
        Type ageType = age.GetType();
        Console.WriteLine(ageType);
    }
}

```

System.Int32

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

```

class MyClass
{
    static void Main()
    {
        int a = 1, b = 2;
        change(ref a, ref b);
        Console.WriteLine("a=" + a + ", b=" + b);
        Console.ReadLine();
    }

    private static void change(ref int a, ref int b)
    {
        int c = a;
        a = b;
        b = c;
    }
}

```

2 1

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

```

internal class A
{
    public A() { } //1
    public int A() { } //2
    public A(int someI) { } //3
    public A(A somA) { } //4
}

```

2

34. Сколько аргументов может быть задано при вызове конструктора данного класса?

```

class Motorcycle
{
    private int driverIntensity;
    private string driverName;
    ссылка: 0
    public Motorcycle(int intensity = 0, string name = "")
    {
        if (intensity > 10)
        {
            intensity = 10;
        }
        driverIntensity = intensity;
        driverName = name;
    }
}

```

Max 2?

35. Почему не удастся создать объект класса A?

```

internal class A
{
    A() { } //1
    A(String st) { } //2
    A(int a) { } //3
    public A(int a, int b) { } //4
}
static void Main(string[] args)
{
    A obj = new A(5);
}

```

Из-за уровня защиты A(int)

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

```

class A
{
    static A() { Console.WriteLine("A static "); }
    public A() { Console.WriteLine("A "); }
}

class Program
{
    static void Main()
    {
        new A();
    }
}

```

Сначала статик, потом публик

37. Какая строка приведенного далее класса вызовет ошибку компиляции?

```
class Points
{
    public readonly int a =10;
    public static readonly Int32 b = new Int32();
    public static string c = "New";
    private int d;

    public Points()
    {
        c = "Method"; //1
        a = 20; //2
        b = 30; //3
    }
}
```

3, можно только в статическом конструкторе