1. **Назовите принципы ООП. Поясните каждый из них.**
2. **Назовите класс .NET, от которого наследуются все классы.**

System.Object

1. **Охарактеризуйте открытые методы System.Object.**

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 вызывающему 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.

1. **Охарактеризуйте закрытые методы System.Object.**

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

protected object MemberwiseClone()

Один старый опытный кодер сказал мне, что этот метод используется для копирования объектов, при котором не копируется объект, а копируются его члены. Если честно, то я ему не верю, т.к. метод возвращает значение типа object. Этот метод скорее создает копию объекта, содержащую ссылки на другие объекты. С этой штукой я пока не  
экспериментировал и за базар не отвечаю. Как-нибудь, я разберусь с этим, напишу.

1. **Приведите пример определения класса.**

public class MyClass { }

1. **Какие ключевые слова можно использовать при определении класса?**

public, private, protected, internal

1. **В чем отличие между объектом и классом?**

Объект – экземпляр класса.

1. **Что такое конструктор? Когда вызывается конструктор?**

Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

1. **Перечислите свойства конструктора?**

На конструкторы накладываются следующие ограничения: 1. Конструктор не может иметь возвращаемого значения даже void 2. Как следствие 1 нельзя использовать оператор return() 3. Конструкторы нельзя объявлять виртуальными.

1. **Что такое деструктор (destructor) ?**

В языке C# имеется возможность определить метод, который будет вызываться непосредственно перед окончательным уничтожением объекта системой "сборки мусора". Такой метод называется ***деструктором*** и может использоваться в ряде особых случаев, чтобы гарантировать четкое окончание срока действия объекта. Например, деструктор может быть использован для гарантированного освобождения системного ресурса, задействованного освобождаемым объектом. Ниже приведена общая форма деструктора:

*~имя\_класса () {*

*// код деструктора*

*}*

1. **Что такое this?**

Ключевое слово **this** представляет ссылку на текущий экземпляр класса. В каких ситуациях оно нам может пригодиться? В примере выше определены три конструктора. Все три конструктора выполняют однотипные действия - устанавливают значения полей name и age. Но этих повторяющихся действий могло быть больше. И мы можем не дублировать функциональность конструкторов, а просто обращаться из одного конструктора к другому через ключевое слово this, передавая нужные значения для параметров

class Person

{

    public string name;

    public int age;

    public Person() : this("Неизвестно")

    {

    }

    public Person(string name) : this(name, 18)

    {

    }

    public Person(string name, int age)

    {

        this.name = name;

        this.age = age;

    }

    public void GetInfo()

    {

        Console.WriteLine($"Имя: {name}  Возраст: {age}");

    }

}

В данном случае первый конструктор вызывает второй, а второй конструктор вызывает третий. По количеству и типу параметров компилятор узнает, какой именно конструктор вызывается.

1. **Что будет выведено в результате выполнения**

5 5

7 7

Потому что они ссылаются на одну и ту же ячейку в памяти (поверхностное копирование)

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

* **public**: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
* **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.
* **protected**: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
* **internal**: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
* **protected internal**: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.
* **private protected**: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.

Объявление полей класса без модификатора доступа равнозначно их объявлению с модификатором **private**. Классы, объявленные без модификатора, по умолчанию имеют доступ **internal**.

1. **Опишите модификатор protected internal.**

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

1. **Зачем и как используются ref и out параметры функции?**

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.

1. **Приведите пример необязательных и именованных параметров метода.**

static int mySum(int a, int b = 5, int c = 10) \\ необязательный параметр b и c

{

return a + b + c;

}

**int** sum1 = mySum(a: 3, b: 10);

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

Static, readonly, constant

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

class Person

{

    private string name;

    public string Name

    {

        get

        {

            return name;

        }

        private set

        {

            name = value;

        }

    }

    public Person(string name, int age)

    {

        Name = name;

        Age = age;

    }

}

Через свойства устанавливается доступ к приватным переменным класса. Подобное сокрытие состояния класса от вмешательства извне представляет механизм **инкапсуляции**, который представляет одну из ключевых концепций объектно-ориентированного программирования. (Стоит отметить, что само понятие инкапсуляции имеет довольно много различных трактовок, которые не всегда пересекаются друг с другом) Применение модификаторов доступа типа private защищает переменную от внешнего доступа. Для управления доступом во многих языках программирования используются специальные методы, геттеры и сеттеры. В C# их роль, как правило, выполняют свойства.

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

value. Тип этого параметра определяется типом свойства.

1. **Что такое автоматические свойства?**

public int Age { get; set; }

На самом деле тут также создаются поля для свойств, только их создает не программист в коде, а компилятор автоматически генерирует при компиляции.

В чем преимущество автосвойств, если по сути они просто обращаются к автоматически создаваемой переменной, почему бы напрямую не обратиться к переменной без автосвойств? Дело в том, что в любой момент времени при необходимости мы можем развернуть автосвойство в обычное свойство, добавить в него какую-то определенную логику.

Стоит учитывать, что нельзя создать автоматическое свойство только для записи, как в случае со стандартными свойствами.

1. **Что такое индексаторы класса? Какие ограничения существуют на индексатор?**

Индексаторы позволяют приложению обращаться с объектом класса так, как будто он является массивом. Индексатор во многом напоминает свойство, но в отличие от свойства он принимает в качестве параметра индекс массива. Так как объект класса используется как массив, то в качестве имени класса используется ключевое слово this. Определение индексатора синтаксически выглядит следующим образом:

public float this[int j]

{

get

{ //Возврат необходимых данных

}

Set

{ //Установка необходимых данных

}

}

1. **Что такое перегруженный метод?**

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#.

1. **Что такое partial класс и какие его преимущества?**

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**.

1. **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.

1. **Для чего делают статические классы?**

Класс можно объявлять как static. Статический класс обладает двумя основными свойствами. Во-первых, объекты статического класса создавать **нельзя**. И во-вторых, статический класс должен содержать только статические члены. Статический класс создается по приведенной ниже форме объявления класса, видоизмененной с помощью ключевого слова static.

*static class имя класса { // ...*

Статические классы применяются главным образом в двух случаях. Во-первых, статический класс требуется при создании *метода расширения*. Методы расширения связаны в основном с языком LINQ. И во-вторых, статический класс служит для хранения совокупности связанных друг с другом статических методов

1. **В чем отличие статического поля от экземплярного?**

Статические поля можно вызывать без создания экземпляра класса.

1. **Поясните работу статических конструкторов.**

Статический конструктор, как правило, используется для инициализации компонентов, применяемых ко всему классу, а не к отдельному экземпляру объекта этого класса. Поэтому члены класса инициализируются статическим конструктором до создания каких-либо объектов этого класса

1. **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.

1. **В чем разница между равенством и тождеством объектов?**

Проверка равенства ссылок объектов – проверка на тождество.

1. **Что такое частичные классы и частичные методы?**

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.

**Частичные методы**

Частичные классы могут содержать частичные методы. Таким методы также определяются с ключевым словом **partial**. Причем определение частичного метода без тела метода находится в одном частичном классе, а реализация этого же метода - в другом частичном классе.

Например, изменим выше определенные классы Person. Первый класс:

1. **Что будет выведено на консоль результате выполнения следующего кода:**

System.Int32

1. **Что будет выведено на консоль результате выполнения следующего кода:**

A= 2 b = 1

1. **Пусть задан следующий класс. Какой из конструкторов задан неверно?**

2ой, т.к. конструктор не возвращает значений

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

0, 1 или 2

1. **Почему не удается создать объект класса A?**

У конструктора A(int a){} не задан модификатор доступа, поэтому он приватный и не доступен вне класса A.

1. **Что будет выведено в консоль при выполнении данной программы?**

A static

A

1. **Какая строка приведенного далее класса вызовет ошибку компиляции?**

3, т.к. присваивание статическому полю, доступному только для чтения допускается только в статическом конструкторе класса.