**1. Назовите принципы ООП. Поясните каждый из них.**

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

2) Наследование (процесс, благодаря которому один объект может наследовать свойства другого объекта / иерархии классов)

3) Полиморфизм (способность вызывать метод потомка через экземпляр предка / способность программы идентично использовать объекты с одинаковым интерфейсом без информации о конкретном типе этого объекта)

4) 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.**

public override bool Equals (object? obj) {...} // и возврашает true, если оба объекта равны

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

public override int GetHashCode(); // возвращает хеш-код

person.GetType() // возвращает тип объекта

i.ToString() // возвращает строку

**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), о котором подробнее см. главу 5.
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. Если ваши классы должны поддерживать полную копию (deep copy), которая действительно включает копии объектов, на которые она ссылается, то вам нужно реализовать интерфейс ICloneable и самому вручную производить клонирование или копирование.

**5. Приведите пример определения класса.**

[ атрибуты ] [ спецификаторы ]

Class имякласса [ предок ]

{тело-класса }

class Book

{

public string name;

public string author;

}

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

New

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

Класс – шаблон, объект – сущность, собранная по шаблону

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

Конструкторы —это специальные методы, позволяющие корректно инициализировать новый экземпляр типа. Вызывается при создании объекта.

**9. Перечислите свойства конструктора?**

1) имя = имя типа (класса)

2) не имеет возвращаемого значения

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 (seald и abstract) 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)?**

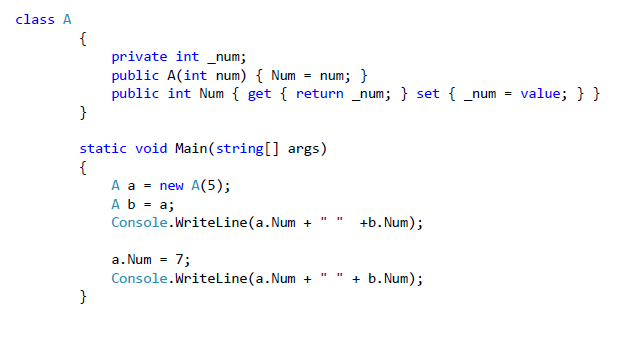
~имя\_класса () {код деструктора}

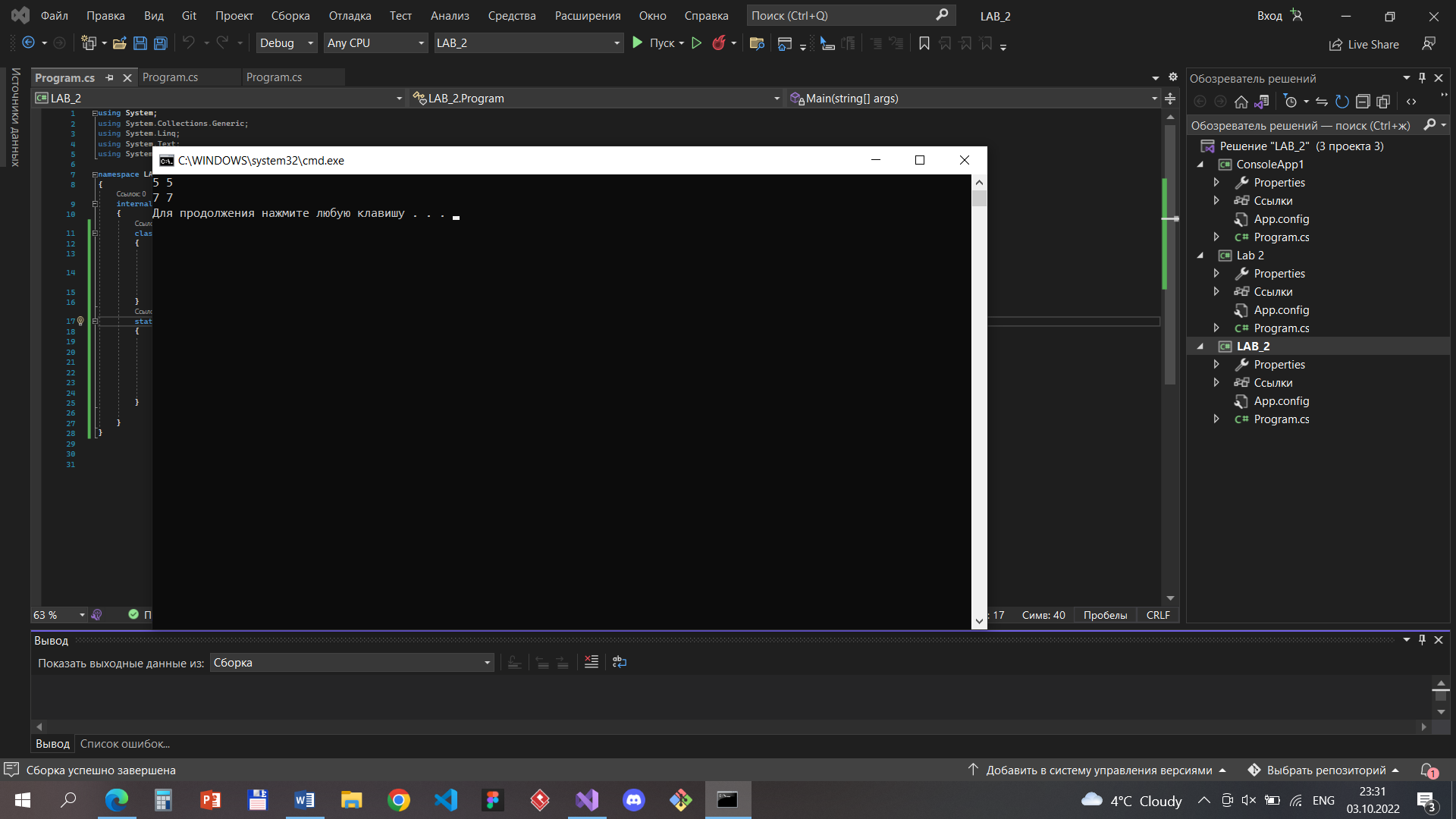
Дестру́ктор — специальный метод класса, служащий для деинициализации объекта. Вызывается перед окончательным уничтожением объекта системой "сборки мусора".

**11. Что такое this?**

Ключевое слово, обеспечивающее доступ к текущему экземпляру класса

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





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

Для классов:

* Private (только в рамках своего класса или структуры)
* Private protected (из любого места класса или производных класса (классы только в той же сборке))
* Protected (из любого места класса или производных класса (классы могут располагаться в разных сборках))
* Protected internal (в любом месте сборки и в производных в других сборках)
* Internal (в любом месте сборки)
* Public (везде)

**14. Опишите модификатор protected internal.**

**Protected** : такой компонент класса доступен из любого места в своем классе или в производных классах. При этом производные классы могут располагаться в других сборках. **internal** : компоненты класса или структуры доступен из любого места кода в той же сборке, однако он недоступен для других программ и сборок.

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

Ref – параметры ссылки (вызов по значению меняет на вызов по ссылке)

Out – выходные параметры ссылки (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.**

Для необязательных параметров нам необходимо объявить значение по умолчанию. После необязательных параметров все последующие параметры также должны быть необязательными

void PrintPerson(string name, int age = 1, string company = "Undefined")

{ Console.WriteLine($"Name: {name} Age: {age} Company: {company}"); }

PrintPerson("Tom", 37, "Microsoft");  // Name: Tom  Age: 37  Company: Microsoft

PrintPerson("Tom", 37); // Name: Tom  Age: 37  Company: Undefined

PrintPerson("Tom"); // Name: Tom Age: 1 Company: Undefined

Именованные параметры помогают передавать аргументы не по позиции.

void PrintPerson(string name, int age = 1, string company = "Undefined")

{ Console.WriteLine($"Name: {name} Age: {age} Company: {company}"); }

PrintPerson ("Tom", company: "Microsoft", age: 37); // Name: Tom Age: 37 Company: Microsoft

PrintPerson (age:41, name: "Bob"); // Name: Bob Age: 41 Company: Undefined

PrintPerson (company: "Google", name: "Sam"); // Name: Sam Age: 1 Company: Google

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

class Person

{

public const string type = "Person";

public string name = "Undefined";

public readonly int age; // запись при объявлении или в виде конструктора

public void Print() => Console.WriteLine($"{type}: {name} - {age}");

}

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

**Инкапсуляция** – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали  
реализации от пользователя.  
  
Инкапсуляция неразрывно связана с понятием интерфейса класса. По сути, всё то, что не входит в интерфейс, инкапсулируется в классе.

Person person = new Person();

Console.WriteLine(person.Age);  // 1

// изменяем значение свойства

person.Age = 37;

Console.WriteLine(person.Age);  // 37

// пробуем передать недопустимое значение

person.Age = -23;               // Возраст должен быть в диапазоне от 1 до 120

Console.WriteLine(person.Age);  // 37 - возраст не изменился

class Person

{

    int age = 1;

    public int Age

    {

        set

        {

            if (value < 1 || value > 120)

                Console.WriteLine("Возраст должен быть в диапазоне от 1 до 120");

            else

                age = value;

        }

        get { return age; }

    }

}

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

set {действия, выполняемые при установке значения свойства}

value

**20. Что такое автоматические свойства?**

Свойства, в которых компилятор автоматически реализует методы для правильного возвращения значения из поля и назначения значения полю

public string Name { get; set; }

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

Свойства с параметрами, которые позволяют индексировать объекты таким же способом, как массив или коллекцию

public Person this[int index]

{ get => personal[index];

set => personal[index] = value; }

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?**

**Перегрузка** **методов** – это объявление в классе **методов** с одинаковыми именами при этом с различными параметрами. Имея некий **метод**, чтобы его **перегрузить**, другой **метод** с таким же именем должен отличаться от него количеством параметров и/или типами параметров. Отличия только типами возвращаемых значений **методами** недостаточно для **перегрузки**, но если **методы** отличаются параметрами, тогда **перегружаемые** **методы** могут иметь и различные типы возвращаемых значений.

23. Что такое partial класс и какие его преимущества?

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.

Но есть ряд ограничений: нельзя модификаторы, только тип Void

Стоит отметить, что по умолчанию к частичным методам применяется ряд ограничений:

* Они не могут иметь модификаторы доступа
* Они имеют тип **void**
* Они не могут иметь out-параметры
* Они не могут иметь модификаторы virtual, override, sealed, new или extern

24. Что такое анонимный тип в C#?

**Анонимные** **типы**, их определение и использование в языке программирования **C#**, особенности компиляции **анонимных** **типов**. ... **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.

25. Для чего делают статические классы?

Компилятор может проверить отсутствие случайно добавленных членов экземпляра, что гарантирует невозможность создания экземпляров таких классов. Статические классы запечатаны поэтому их нельзя наследовать.

26. В чем отличие статического поля от экземплярного?

Поле – это переменная объявленная внутри класса или структуры, член типа. Вне типа поле нельзя объявить и создать.

**В классах или структурах встречаются в двух видах:**

* поля экземпляров;
* статические поля.

*Поля экземпляров*отличаются от *статических полей* тем, что их значение различно для разных экземпляров типа, значения статических полей в разных экземплярах типа одинаковы.

Назначение полей – хранить меняющиеся данные.

27. Поясните работу статических конструкторов.

1. 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.

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

1. 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?

1. Понятие и знак "равно" используют:
2. а) для записи верных числовых равенств, например, 3+2=5.
3. б) для записи уравнений, т. е. соотношений, содержащих переменные
4. величины, причем такое соотношение может быть верным при
5. некоторых значениях переменных, называемых решениями (или
6. корнями) уравнения. Например, 3х=9 верно при х=3.
8. Понятие "тождество" (или тождественное равенство, обозначают
9. тремя черточками) означает равенство, содержащее переменные
10. величины, причем это равенство должно быть верным при всех
11. (допустимых) значениях переменных, например, 3(х+у) =3х+3у.

30. Что такое частичные классы и частичные методы?

1. 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.
2. Есть несколько ситуаций, когда удачно разбить определение класса на несколько файлов:
3. 1. При работе над большим проектом, разбиение класса на несколько файлов позволяет нескольким программистам работать над ним одновременно.
4. 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.
6. Для такого разбития класса используется модификатор partial.

9. 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:

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

33. Пусть задан следующий класс.

Какой из конструкторов задан неверно?

34. Пусть задан следующий класс.

Сколько аргументов может быть задано при вызове конструктора данного класса?

35. Почему не удается создать объект класса A?

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

37. Какая строка приведенного далее класса вызовет ошибку компиляции?

**Инкапсуляция -**скрытие некоторых моментов реализации класса от других частей программы.