Вопросы:

1. 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

Абстракция подразумевает разделение и  
независимое рассмотрение интерфейса  
и реализации

Полиморфизм - способность вызывать метод потомка  
через экземпляр предка

2. System.Object

3. ToString()

Метод ToString() возвращает символьную строку, содержащую описание того объекта, для которого он вызывается.

GetHashCode()

Этот метод используется, когда объект помещается в структуру данных, известную как карта (map), которая также называется [хеш-таблицей](https://professorweb.ru/my/csharp/charp_theory/level12/12_5.php) или словарем.

Equals() и ReferenceEquals()

По умолчанию метод Equals (object) определяет, ссылается ли вызывающий объект на тот же самый объект, что и объект, указываемый в качестве аргумента этого метода, т.е. он определяет, являются ли обе ссылки одинаковыми.

GetType()

Этот метод возвращает экземпляр класса, унаследованный от System.Type.

4. Finalize()

Назначение этого метода в C# примерно соответствует деструкторам С++, и он вызывается при сборке мусора для очистки ресурсов, занятых ссылочным объектом. Реализация Finalize() из Object на самом деле ничего не делает и игнорируется сборщиком мусора. Обычно переопределять Finalize() необходимо, если объект владеет неуправляемыми ресурсами, которые нужно освободить при его уничтожении.

Clone()

Этот метод создает копию объекта и возвращает ссылку на эту копию (а в случае типа значения — ссылку на упаковку)

5. class Program

{

static void Main(string[] args)

{

var m = Environment.Version;

Console.WriteLine("Тип m: "+m.GetType());

string s = m.ToString();

Console.WriteLine("Моя версия .NET Framework: " + s);

Version v = (Version)m.Clone();

Console.WriteLine("Значение переменной v: "+v);

Console.ReadLine();

}

}

6. Атрибуты и спецификаторы (public, private, static).

7. Класс – это некоторое абстрактное понятие - шаблон, по которому определяется форма объекта  
Объект – это физическая реализация класса(шаблона).

8. Конструкторы — это специальные методы, позволяющие корректно инициализировать новый экземпляр типа. Создание экземпляра объекта ссылочного типа  
1) выделяется память для полей данных экземпляра  
2) инициализируются служебные поля  
3) вызывается конструктор экземпляра, устанавливающий исходное состояние нового объекта. Память всегда обнуляется до вызова конструктор экземпляра типа. Любые поля, не задаваемые конструктором явно, гарантированно содержат 0 или null.

9. 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. **Деструкторы** вызывается непосредственно перед окончательным  
уничтожением объекта системой "сборки мусора",  
чтобы гарантировать четкое окончание срока действия  
объекта.

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

12. 5 5 7 7

13. public, private, protected, internal

14. **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

15.

Модификатор параметра ref принудительно организует вызов по ссылке, а не по значению. Этот модификатор указывается как при объявлении, так и при вызове метода.

Модификатор параметра out подобен модификатору ref, за одним

16. Именованные

class Program

{

static int mySum(int a, int b = 5, int c = 10)

{

return a + b + c;

}

static void Main()

{

// Использование именованных аргументов

// при вызове метода

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

Console.WriteLine("Sum1 = " + sum1);

Console.ReadLine();

}

}

Необязательные

class Program

{

// Аргументы b и с указывать при вызове необязательно

static int mySum(int a, int b = 5, int c = 10)

{

return a + b + c;

}

static void Main()

{

int sum1 = mySum(3);

int sum2 = mySum(3,12);

Console.WriteLine("Sum1 = "+sum1);

Console.WriteLine("Sum2 = "+sum2);

Console.ReadLine();

}

17. private static int size;

private readonly int objHash = size + 1;

private const string month = "Октябрь";

18. Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки (предоставляет инкапсуляцию).

public int Hour

{

get => hour;

set

{

if(value >= 0 && value < 24)

hour = value;

else

throw new Exception("Неверный формат времени.");

}

}

19. Value

20. 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. У таких свойств, у их аксессоров отсутствует тело.

21. Индексаторы – это синтаксический инструмент языка C#, который позволяет индексировать внутренние коллекции типа. Индексаторы позволяют индексировать объекты и обращаться к данным по индексу. Фактически с помощью индексаторов мы можем работать с объектами как с массивами. По форме они напоминают свойства со стандартными блоками get и set, которые возвращают и присваивают значение.

22. 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 (method overloading). И в языке C# мы можем создавать в классе несколько методов с одним и тем же именем, но разной сигнатурой.

23. 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. Для этого они определяются с ключевым словом partial.

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

24. Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса. Анонимный тип определяется с помощью ключевого слова var и инициализатора объектов.

25. Статические классы объявляются с модификатором static и могут содержать только статические поля, свойства и методы. Иногда требуется определить такой член класса, который будет использоваться независимо от всех остальных объектов этого класса. Как правило, доступ к члену класса организуется посредством объекта этого класса, но в то же время можно создать член класса для самостоятельного применения без ссылки на конкретный экземпляр объекта.

26.

27. Статический конструктор используется для инициализации любых [статических](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/static) 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. 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. Метод Equals принимает в качестве параметра объект любого типа, который мы затем приводим к текущему, если они являются объектами одного класса. Затем сравниваем по именам. Если имена равны, возвращаем true, что будет говорить, что объекты равны.

30. 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 классы могут содержать частичные методы. Таким методы также опреляются 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.

31.

****

System.Int32

32.

****

a=2, b=1

33.

****

**Какой из конструкторов задан неверно?**

2, 4(не уверен про 4)

**34.**

****

**Сколько аргументов может быть задано при вызове конструктора**

**данного класса?**

2

35.

****

'Program.A.A(int)" недоступен из-за его уровня защиты.

36.

****

A static

A

37.

****

b=30; т.к. присвоение значения доступному только для чтения статическому полю допускается только в статическом конструкторе и в инициализаторе переменных.

Time.cs

using System;

namespace OOP\_Lab3

{

partial class Time

{

// ПОЛЯ ====================================================

private static int size;

private readonly int objHash = size + 1;

private const string month = "Октябрь";

private int hour;

private int minute;

private int second;

// СВОЙСТВА ================================================

public int Hour

{

get => hour;

set

{

if(value >= 0 && value < 24)

hour = value;

else

throw new Exception("Неверный формат времени.");

}

}

public int Minute

{

get => minute;

set

{

if (value >= 0 && value < 59)

minute = value;

else

throw new Exception("Неверный формат времени.");

}

}

public int Second

{

get => second;

set

{

if (value >= 0 && value < 59)

second = value;

else

throw new Exception("Неверный формат времени.");

}

}

public string Month { get; }

// КОНСТРУКТОРЫ ============================================

static Time()

{

Console.WriteLine("Начата работа с классом \"Time\_Class\"");

}

public Time()

{

Hour = 0;

Minute = 0;

Second = 0;

Time.size++;

}

public Time(int hour = 0, int minute = 0, int second = 0)

{

this.Hour = hour;

this.Minute = minute;

this.Second = second;

Time.size++;

}

private Time(int second)

{

hour = 0;

minute = 0;

this.Second = second;

Time.size++;

}

// МЕТОДЫ ===================================================

public override bool Equals(object obj)

{

Time tmp = obj as Time;

if (tmp != null)

{

if (this.Hour == tmp.Hour && this.Minute == tmp.Minute && this.Second == tmp.Second)

return true;

else

return false;

}

throw new Exception("Невозможно преобразовать объект к типу Time\_Class.");

}

public override int GetHashCode() => objHash;

public override string ToString() => $"Hours: {Hour}, Minutes: {Minute}, Seconds: {Second}";

public static void HoursSum(Time objOne, Time objTwo, out Time objOut) // ничего лучше что-то не придумалось (главное, что суть ясна)

{

Time result = new Time(objOne.Hour + objTwo.Hour);

objOut = result;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OOP\_Lab3

{

class Program

{

static void Main(string[] args)

{

Time one = new Time(12);

Time two = new Time(23);

Time three = new Time(13,40,50);

Time four = new Time(hour: 19,second: 12);

Time five = new Time(second: 36);

Console.WriteLine(four);

Time[] timeArray = { one, two, three, four, five};

Time.GetInfo();

var myView = new { hour = 1, minute = 40, second = 59 };

Console.WriteLine("Введите число: ");

int number = int.Parse(Console.ReadLine());

foreach (Time item in timeArray)

{

if(item.Hour == number)

Console.WriteLine(item);

}

Time.HoursSum(one, two, out five);

one.Equals(two);

for (int i=0; i<5; i++)

{

if (timeArray[i].Hour<=12)

{

Console.Write("Утро - ");

Console.WriteLine(timeArray[i]);

}

else if(timeArray[i].Hour<=18)

{

Console.Write("День - ");

Console.WriteLine(timeArray[i]);

}

else if (timeArray[i].Hour <= 22)

{

Console.Write("Вечер - ");

Console.WriteLine(timeArray[i]);

}

else if (timeArray[i].Hour <= 24)

{

Console.Write("Ночь - ");

Console.WriteLine(timeArray[i]);

}

}

Console.ReadKey();

}

}

}

TimePart.cs

using System;

namespace OOP\_Lab3

{

partial class Time

{

public static void GetInfo() =>

Console.WriteLine("Создано экземпляров класса: {0}", size);

}

}