1) Четыре основных принципа объектно-ориентированного программирования следующие.

Абстракция. Разделение программы на объекты

Инкапсуляция. Скрытие данных при помощи private или protect

Наследование. Возможность создания новых абстракций на основе существующих

Полиморфизм. Возможность переопределния, полученного от предков

2) System.Object. – Базовый класс

3) Все классы в .NET, даже те, которые мы сами создаем, а также базовые типы, такие как **System.Int32**, являются неявно производными от класса Object. Даже если мы не указываем класс Object в качестве базового, по умолчанию неявно класс Object все равно стоит на вершине иерархии наследования. Поэтому все типы и классы могут реализовать те методы, которые определены в классе 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 GetHashCodeO 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 (см. о них главу 16) 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) 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.

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) Класс представляет собой инкапсуляцию данных и методов их обработки. Это справедливо для любого объектно-ориентированного языка и отличаются они в этом плане лишь типами тех данных, которые можно хранить в виде членов, а также возможностями классов. Синтаксис определения классов на С#, прост, особенно если вы программируете на C++ или Java. Поместив перед именем вашего класса ключевое слово class, вы вставляете члены класса, заключенные в фигурные скобки, например:

class Employee {

private long employeeld; }

6) Модификаторы доступа, static, partial

7) Можно сказать, что класс – это описание какого-то типа данных. Это некий абстрактный шаблон (набор правил), по которому мы можем создавать объекты (переменные, того типа данных, который описывает класс). Объект – это созданный экземпляр класса. Например, есть класс «Студент», который характеризует среднестатистического студента (с фамилией, именем, возрастом и т.п.), а есть конкретный студент Иванов Иван 20-ти лет.

8) Конструктор — это специальный метод, который имеет имя, совпадающее с именем класса, и вызывается при создании экземпляра объекта совместно с оператором new. Результатом работы этого метода всегда является экземпляр класса. Следовательно, конструктор класса Cat называется Cat (). В результате работы этого конструктора будет создан новый объект класса Cat. Цель конструктора — правильно инициализировать объект перед его использованием.

9) Вызывается при инициализации класса

10) Деструктор, это метод который вызывается для гарантированного освобождения памяти под объект. В деструкторе указываются те действия, которые необходимо выполнить прежде чем уничтожить объект. Общая форма деструктора имеет вид:~ имя\_класса () {// код деструктора} где имя\_класса – имя конкретного класса. У деструктора отсутствует тип, который возвращается и список параметров.

11) В языке C# имеется ключевое слово this, которое обеспечивает доступ к текущему экземпляру класса. Одно из возможных применений ключевого слова this состоит в том, чтобы разрешать неоднозначность контекста, которая может возникнуть, когда входящий параметр назван так же, как поле данных данного типа.

12) 5+5

7+5

[\\\или](file:///\\\или) ошибка??

13) - public – общедоступный член класса.

- private – член класса доступен только внутри данного класса.

- protected – член класса доступен только внутри данного класса и внутри классов, производных от данного.

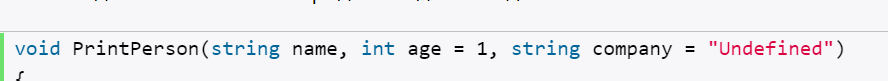
- internal – член класса доступен только внутри данной сборки (программы).

14) Комбинация ключевых слов protected internal является модификатором доступа к члену. Доступ к членам с модификатором доступа protected internal может осуществляться из текущей сборки или типов, которые являются производными от содержащего класса.

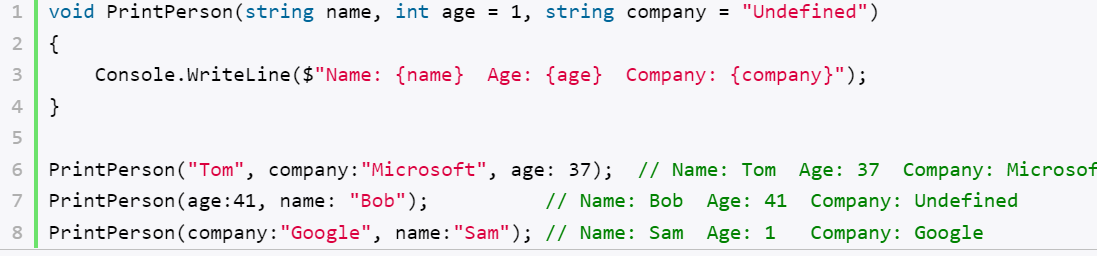
15) В отличии от функций С, при передаче методу параметров по адресу, необходимо указывать ключевое слово **ref** или **out**. Эти ключевые слова сообщают компилятору, что адреса параметров функции совпадают с адресами переменных, передаваемых в качестве параметров.

16) **Параметры** позволяют передать в метод некоторые входные данные. Параметры определяются через заятую в скобках после названия метода в виде.

По умолчанию при вызове метода необходимо предоставить значения для всех его параметров. Но C# также позволяет использовать необязательные параметры. Для таких параметров нам необходимо объявить значение по умолчанию. Также следует учитывать, что после необязательных параметров все последующие параметры также должны быть необязательными



В предыдущих примерах при вызове методов значения для параметров передавались в порядке объявления этих параметров в методе. То есть аргументы передавались параметрам **по позиции**. Но мы можем нарушить подобный порядок, используя именованные параметры:



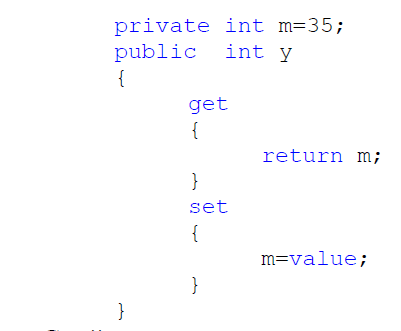
17) static int count = 5;

Public readonly string name = “Aleksey”

Const int telephone\_number = 2343454;

18) Свойства в С# состоят из объявления поля и методов-аксессоров для работы с этим полем.

Эти методы- аксессоры называются получатель (get) и установщик (set). Например, простейшее свойство **y**, работающее с полем **m,** можно представить следующим образом:



Свойство, определяется, так же как и поле, но после имени свойства идет блок кода, включающий в себя два метода get и set. Код этих методов может быть сколь угодно сложным, но в нашем случае это всего лишь один оператор. Аксессор **get** всегда возвращает значение того типа, который указан в определении свойства. Аксессор **set** всегда принимает в качестве параметра переменную **value**, которая передается ему неявно. Один из аксессоров может

быть опущен, в этом случае мы получаем поле только для чтения или только для записи.

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. Структура объявления Автоматического свойства: [модификатор доступа] [тип] [имя\_свойства] {get; set;}

21) Индексатор – это средство языка C#, позволяющее индексировать объект так как массив с помощью прямоугольных скобок []. С помощью индексаторов можно реализовывать собственные специализированные массивы, на которые могут накладываться различные ограничения.

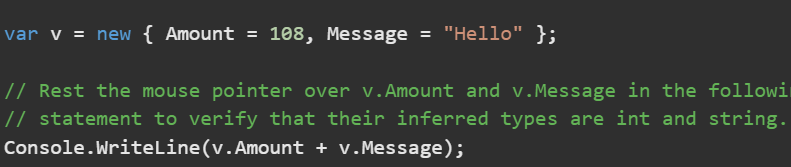
Индексаторы позволяют приложению обращаться с объектом класса так, как будто он является массивом. Индексатор во многом напоминает свойство,

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**.

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).

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.

24) Анонимные типы в C# - это типы, которые не имеют имени или, можно сказать, создание новых типов без их определения. Он представлен в C# 3.0. Это временный тип данных, который выводится на основе данных, вставляемых в инициализатор объекта. Или, другими словами, анонимный тип обеспечивает эффективный способ объединения набора объектов только для чтения в один объект без какого-либо явного типа.



25) В с# статический класс играет роль пространства имен для глобальных переменных. Глобальных переменный в принципе в C# нет, но можно объявлять переменные как поля в статическом классе.

26) 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. Например: Организация содержит 3 типа менеджеров. Нам нужно создать три разных класса.

27) Он вызывается автоматически перед созданием первого экземпляра или ссылкой на любые статические члены.

28) Различают два типа клонирования: поверхностное (shallow) и глубокое (deep). При поверхностном клонировании копируется сам объект. Все значимые поля клона получают значения, совпадающие со значениями полей объекта; все ссылочные поля клона являются ссылками на те же объекты, на которые ссылается и сам объект. При глубоком клонировании копируется вся совокупность объектов, связанных взаимными ссылками.

29)В одном случае необходимо приведение типов, а в другом нет. (double с int)

30)Один общий класс