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

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) Назовите класс .NET, от которого наследуются все классы.**

System.Object. – Базовый класс

**3) Охарактеризуйте открытые методы System.Object.**

### ToString()

* **Описание**: Возвращает строковое представление текущего объекта.
* **Использование**: Обычно переопределяется для предоставления более информативной информации о состоянии объекта.
* **Пример**:

csharp

Copy

public override string ToString()

{

return $"Person: {Name}, Age: {Age}";

}

**2. Equals(object obj)**

* **Описание**: Определяет, равен ли текущий объект другому объекту.
* **Использование**: Переопределяется для определения специфичных правил равенства объектов.
* **Пример**:

csharp

Copy

public override bool Equals(object obj)

{

if (obj is Person other)

{

return this.Name == other.Name && this.Age == other.Age;

}

return false;

}

**3. GetHashCode()**

* **Описание**: Возвращает хэш-код для текущего объекта.
* **Использование**: Переопределяется, если Equals переопределен, чтобы гарантировать, что равные объекты имеют одинаковый хэш-код.
* **Пример**:

csharp

Copy

public override int GetHashCode()

{

return HashCode.Combine(Name, Age);

}

**4. GetType()**

* **Описание**: Возвращает объект Type, представляющий текущий экземпляр объекта.
* **Использование**: Полезно для получения информации о типе объекта в рантайме.

**4)** **Охарактеризуйте закрытые методы System.Object.**

В классе System.Object нет явно определенных закрытых методов, доступных для использования. Однако класс System.Object включает в себя некоторые методы, которые могут быть не доступны для переопределения или прямого вызова, так как они предназначены для внутреннего использования или управляются системой. Например:

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) Приведите пример определения класса.**

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) Какие ключевые слова можно использовать при определении класса?**

Модификаторы доступа, static, partial

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

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) Что такое конструктор? Когда вызывается конструктор?**

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) Перечислите свойства конструктора?**

Конструкторы — это специальные методы класса, которые вызываются при создании объекта. Вот основные свойства конструкторов:

*Название*:

* + Конструктор имеет то же имя, что и класс, в котором он объявлен.

*Отсутствие возвращаемого типа:*

* + Конструкторы не имеют возвращаемого типа, даже void.

*Автоматический вызов:*

* + Конструкторы вызываются автоматически при создании объекта.

*Перегрузка:*

* + В классе может быть несколько конструкторов с разными параметрами (перегрузка).

*Инициализация:*

* + Конструкторы используются для инициализации объектов и их атрибутов.

*Дефолтный конструктор:*

* + Если не объявлен ни один конструктор, компилятор создает дефолтный конструктор, который не принимает параметров.

*Конструктор копирования:*

* + Специальный конструктор, который создает новый объект как копию существующего.

*Инициализация через список инициализации:*

* + Конструкторы могут использовать список инициализации для инициализации членов класса до выполнения тела конструктора.

*Инициализация статических членов:*

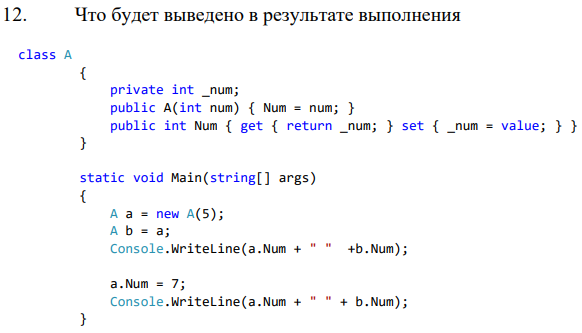
* + Конструкторы могут инициализировать статические члены класса, которые принадлежат классу, а не конкретному объекту.

**10) Что такое деструктор (destructor) ?**

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) Что такое this?**

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+7

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

- 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) Опишите модификатор protected internal.**

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.

- protected – член класса доступен только внутри данного класса и внутри классов, производных от данного. - internal – член класса доступен только внутри данной сборки (программы).

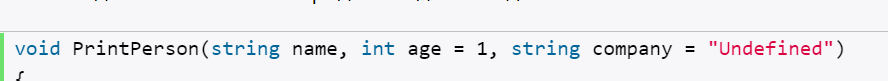
**15) Зачем и как используются ref и out параметры функции?**

Эти ключевые слова сообщают компилятору, что адреса параметров функции совпадают с адресами переменных, передаваемых в качестве параметров. Любое изменение значения параметров в этом случае приведет к изменению и переменных вызывающего кода. Рекомендуется для входного параметра использовать ключевое слово ref, а для выходного параметра ключевое слово out, так как параметр функции с ключевым словом ref должен быть обязательно проинициализирован перед вызовом функции. При вызове методов указание ключевых слов ref и out обязательно.

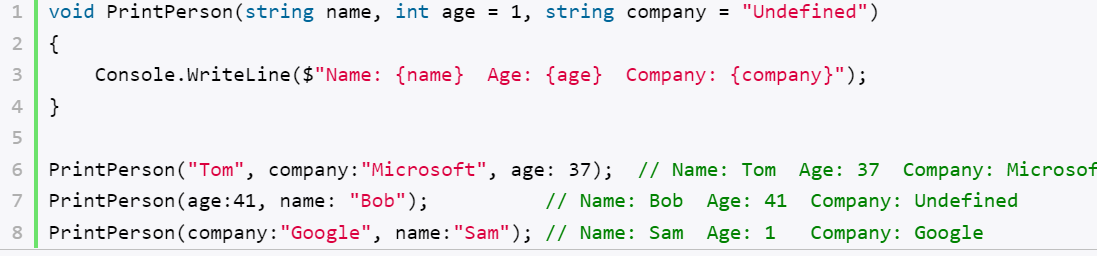
**16) Приведите пример необязательных и именованных параметров метода.**

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

static int count = 5;

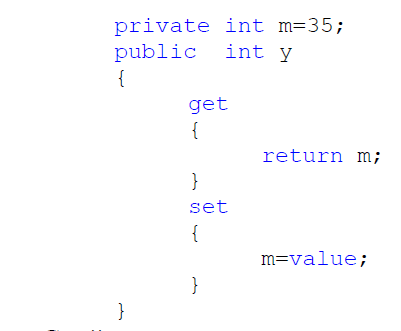
Public readonly string name = “Aleksey”

Const int telephone\_number = 2343454;

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

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

value

**20) Что такое автоматические свойства?**

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) Что такое индексаторы класса? Какие ограничения существуют на индексатор?**

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**.

public float this[int j] { get { //Возврат необходимых данных } set { //Установка необходимых данных } }

**22) Что такое перегруженный метод?**

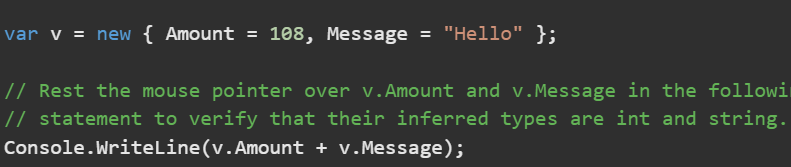
Иногда возникает необходимость создать один и тот же метод, но с разным набором параметров. И в зависимости от имеющихся параметров применять определенную версию метода. Такая возможность еще называется **перегрузкой методов** (method overloading).

**23) 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.

**24) Что такое анонимный тип в C#?**

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) Для чего делают статические классы?**

Статические классы в C# обеспечивают удобный способ организации кода, предоставляя методы и свойства, которые могут быть использованы без создания экземпляров.

**26) В чем отличие статического поля от экземплярного?**

* **Экземплярные поля**: уникальны для каждого объекта, используются для хранения состояния конкретного экземпляра.
* **Статические поля**: общие для всех экземпляров класса, используются для хранения данных, которые должны быть доступны всем объектам.

**27) Поясните работу статических конструкторов.**

Статические конструкторы в C# — это специальные конструкторы, которые используются для инициализации статических данных или выполнения каких-либо действий, которые должны произойти только один раз для всего класса, а не для его экземпляров.

**Основные характеристики статических конструкторов**

1. **Автоматический вызов**:
   * Статический конструктор вызывается автоматически перед первым использованием класса или перед созданием первого экземпляра класса. Вы не можете вызвать его явно.
2. **Отсутствие параметров**:
   * Статические конструкторы не могут принимать параметры. Они не могут быть перегружены.
3. **Один экземпляр**:
   * В классе может быть только один статический конструктор. Если вы попытаетесь определить несколько статических конструкторов, компилятор выдаст ошибку.
4. **Инициализация статических членов**:
   * Статический конструктор обычно используется для инициализации статических полей или выполнения единовременных операций, таких как настройка статических ресурсов.
5. **Неявное создание**:
   * Статический конструктор создается автоматически и не вызывается в обычном коде, что делает его полезным для настройки класса без необходимости явно вызывать его.

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

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 чем разница между равенством и тождеством объектов?**

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

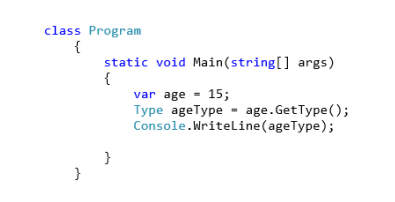
**30) Что такое частичные классы и частичные методы?**

это концепции, используемые в языках программирования, таких как C#, которые позволяют разделять реализацию класса или метода на несколько файлов. Это может быть полезно для организации кода и повышения его читаемости.

Частичный класс — это класс, который может быть разделен на несколько файлов с использованием ключевого слова partial. Все части данного класса объединяются в одну во время компиляции.

Частичный метод — это метод, который может быть объявлен в одной части частичного класса и реализован в другой. Частичные методы также используют ключевое слово partial.

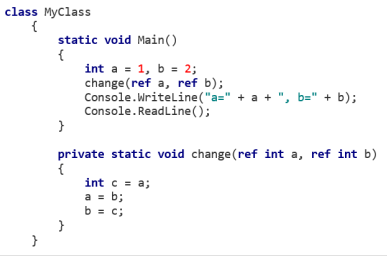
**31) Что будет выведено на консоль результате выполнения следующего кода:**



Ошибка компиляции

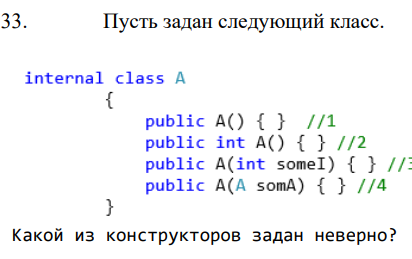
**32) Что будет выведено на консоль рез**

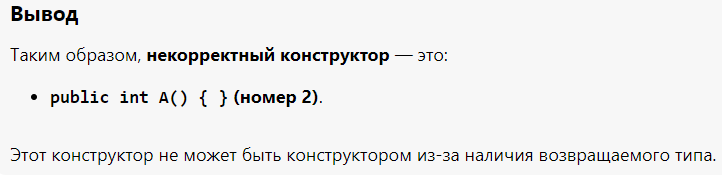
**ультате выполнения следующего кода:**

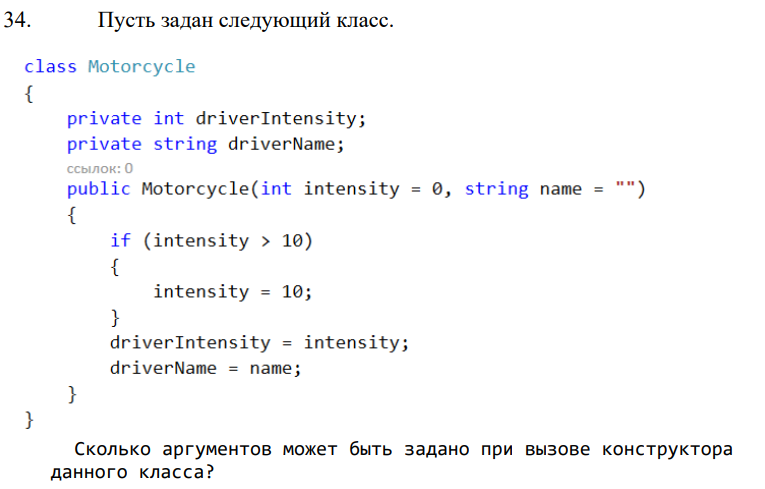


Синтаксическая ошибка

**33) Какой из конструкторов задан неверно?**



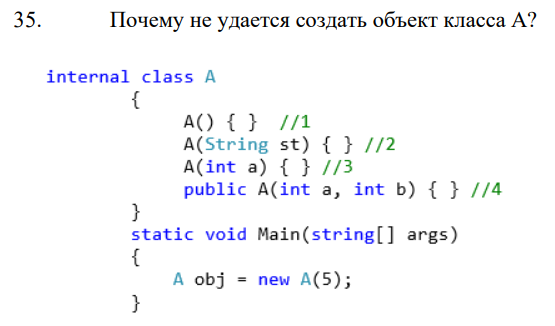




Таким образом, конструктор класса Motorcycle может быть вызван **тремя способами**:

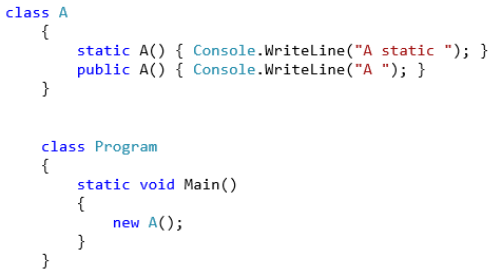
1. Без аргументов.
2. С одним аргументом (первым).
3. С двумя аргументами.

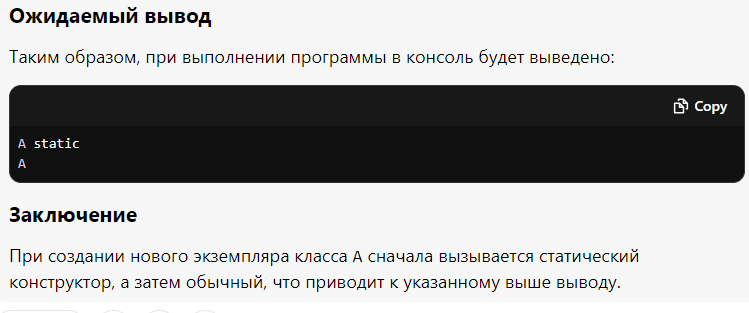
Это значит, что при вызове конструктора можно передать от 0 до 2 аргументов.

35. 

в классе **отсутствует публичный конструктор**, принимающий только один аргумент типа int.

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





**37.** 