### 1. Принципы ООП:

- \*\*Инкапсуляция\*\*: скрытие внутренней реализации объекта и предоставление доступа к его данным через публичные методы (геттеры и сеттеры).

- \*\*Наследование\*\*: возможность одного класса получать свойства и методы другого класса.

- \*\*Полиморфизм\*\*: способность объектов разных классов обрабатываться через общий интерфейс.

- \*\*Абстракция\*\*: представление сложной системы через упрощённый набор значимых характеристик.

### 2. Базовый класс .NET:

Все классы в .NET наследуются от класса `System.Object`.

### 3. Открытые методы `System.Object`:

- \*\*`Equals()`\*\*: сравнивает два объекта на равенство.

- \*\*`GetHashCode()`\*\*: возвращает хэш-код объекта.

- \*\*`GetType()`\*\*: возвращает тип объекта.

- \*\*`ToString()`\*\*: возвращает строковое представление объекта.

### 4. Закрытые методы `System.Object`:

Закрытые методы `System.Object` не могут быть вызваны напрямую

### 5. Пример определения класса:

```csharp

public class Person {

public string Name { get; set; }

public int Age { get; set; }

}

```

### 6. Ключевые слова для определения класса:

- \*\*`public`\*\*, \*\*`private`\*\*, \*\*`protected`\*\*, \*\*`internal`\*\* (модификаторы доступа).

- \*\*`abstract`\*\*, \*\*`sealed`\*\* (ограничение наследования).

- \*\*`static`\*\* (для статических классов).

### 7. Отличие между объектом и классом:

- \*\*Класс\*\* — это шаблон или схема, которая описывает объект.

- \*\*Объект\*\* — это экземпляр класса, который хранит данные и поведение, описанные в классе.

### 8. Конструктор:

- \*\*Конструктор\*\* — это специальный метод, который инициализирует объект при его создании.

- Конструктор вызывается автоматически при создании нового экземпляра класса.

### 9. Свойства конструктора:

- Конструкторы могут быть перегружены.

- Может быть статический конструктор для инициализации статических полей.

- Может быть приватным (чтобы запретить создание объектов).

### 10. Деструктор:

- Деструктор — это метод, который вызывается при уничтожении объекта для очистки ресурсов.

### 11. `this`:

`this` — это ссылка на текущий экземпляр класса, которая позволяет обращаться к его членам внутри метода.

### 12. Результат выполнения программы:

```csharp

class A {

private int \_num;

public A(int num) { Num = num; }

public int Num { get { return \_num; } set { \_num = value; } }

}

static void Main(string[] args) {

A a = new A(5);

A b = a;

Console.WriteLine(a.Num + " " + b.Num); // 5 5

a.Num = 7;

Console.WriteLine(a.Num + " " + b.Num); // 7 7

}

```

Объяснение: так как `a` и `b` ссылаются на один и тот же объект, изменение значения в одном месте влияет на другой.

### 13. Спецификаторы доступа в C#:

- \*\*`public`\*\*: доступен всем.

- \*\*`private`\*\*: доступен только внутри класса.

- \*\*`protected`\*\*: доступен внутри класса и его потомков.

- \*\*`internal`\*\*: доступен внутри одной сборки.

- \*\*`protected internal`\*\*: доступен внутри одной сборки и для потомков класса.

### 14. Модификатор `protected internal`:

Доступен для классов внутри той же сборки и для классов-наследников, даже если они находятся в другой сборке.

### 15. `ref` и `out` параметры:

- \*\*`ref`\*\*: параметр передаётся по ссылке, должен быть инициализирован до вызова метода.

- \*\*`out`\*\*: параметр передаётся по ссылке, но должен быть инициализирован внутри метода.

### 16. Пример необязательных и именованных параметров:

```csharp

public void PrintMessage(string message, int times = 1) {

for (int i = 0; i < times; i++) {

Console.WriteLine(message);

}

}

// Вызов метода

PrintMessage(message: "Hello", times: 3);

```

### 17. Пример полей класса:

```csharp

public class Example {

public static int StaticField;

public const int ConstantField = 10;

public readonly int ReadOnlyField;

}

```

### 18. Свойства класса и инкапсуляция:

Свойства управляют доступом к приватным полям, что помогает инкапсулировать данные:

```csharp

public class Person {

private string \_name;

public string Name {

get { return \_name; }

set { \_name = value; }

}

}

```

### 19. Имя параметра в методе `set` свойства:

Явное имя параметра — это \*\*`value`\*\*.

### 20. Автоматические свойства:

Автоматические свойства упрощают создание свойств без явного определения поля:

```csharp

public string Name { get; set; }

```

### 21. Индексаторы класса:

Индексаторы позволяют обращаться к объектам класса, как к массивам:

```csharp

public class SampleCollection<T> {

private T[] arr = new T[100];

public T this[int index] {

get { return arr[index]; }

set { arr[index] = value; }

}

}

```

### 22. Перегруженные методы:

Перегрузка методов позволяет создавать методы с одним именем, но с разными параметрами:

```csharp

public void Print(string message) { }

public void Print(string message, int times) { }

```

### 23. Partial класс:

\*\*`partial`\*\* класс позволяет разделять реализацию класса на несколько файлов.

### 24. Анонимный тип:

Анонимные типы позволяют создавать объекты без явного определения класса:

```csharp

var person = new { Name = "John", Age = 30 };

```

### 25. Статические классы:

Статические классы содержат только статические члены и не могут быть инстанцированы.

### 26. Статическое поле и экземплярное:

Статическое поле принадлежит классу, а не его экземплярам. Экземплярное поле принадлежит конкретному объекту класса.

### 27. Статические конструкторы:

Статические конструкторы вызываются один раз при первом обращении к статическим членам класса.

### 28. Поверхностное и глубокое копирование:

- \*\*Поверхностное копирование\*\* копирует только ссылки на объекты.

- \*\*Глубокое копирование\*\* копирует сами объекты, создавая их новые экземпляры.

### 29. Равенство и тождество объектов:

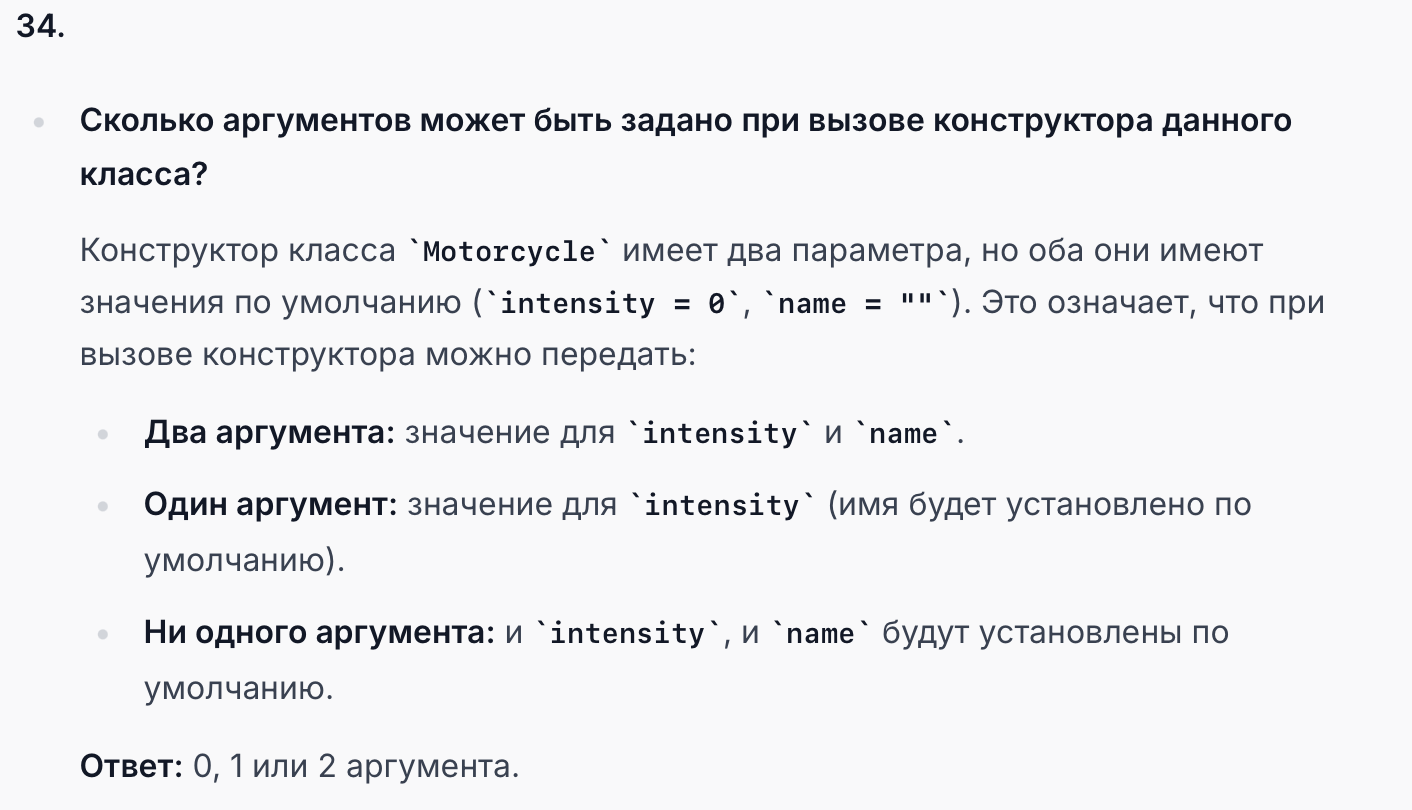
- \*\*Равенство\*\* (`Equals`) проверяет, имеют ли объекты одинаковое значение.

- \*\*Тождество\*\* (`ReferenceEquals`) проверяет, ссылаются ли объекты на одну и ту же область памяти.

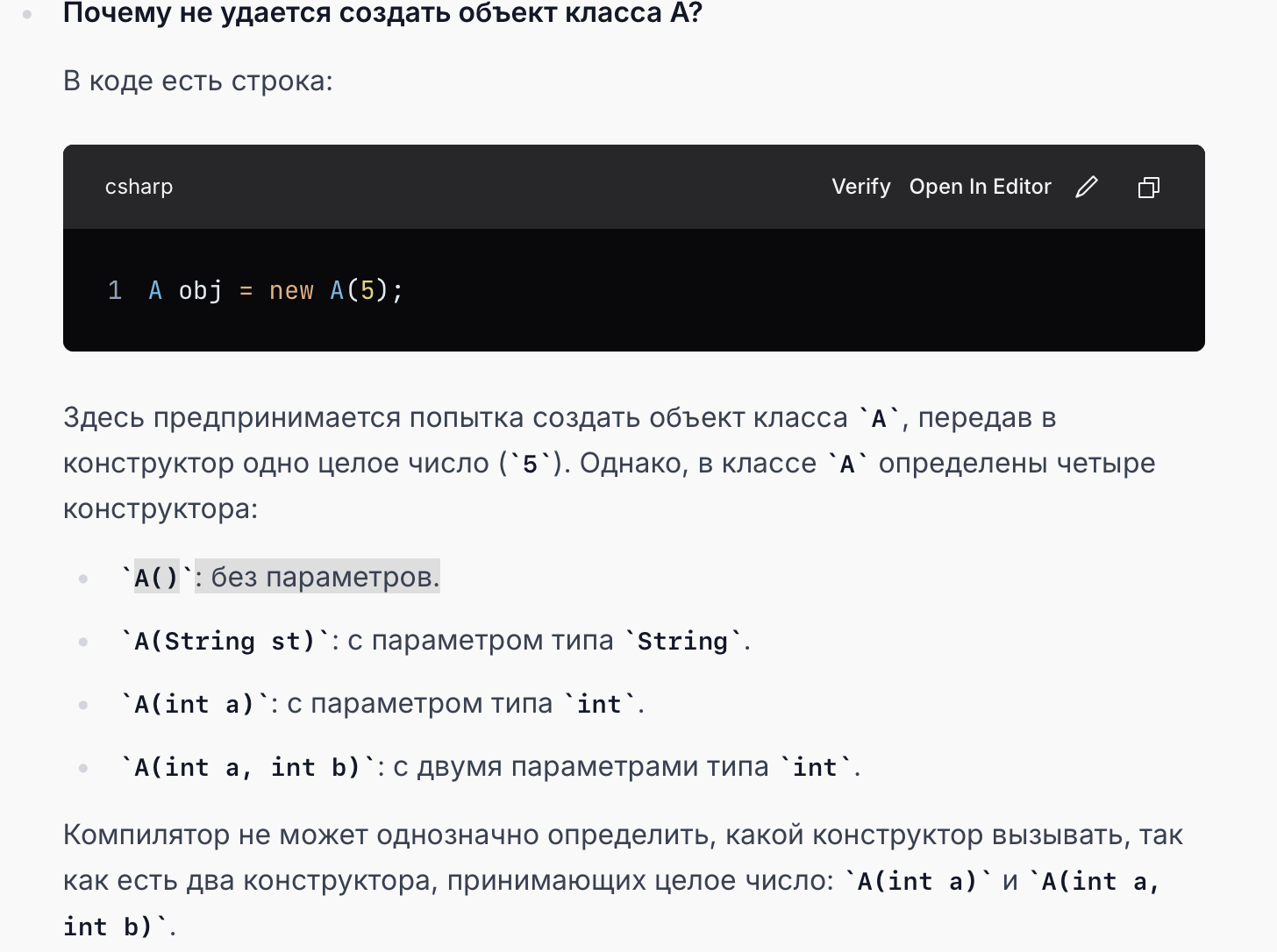
### 30. Частичные классы и методы:

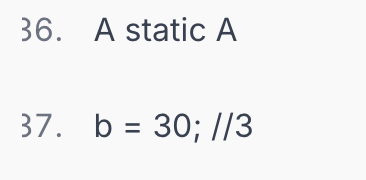
Частичные классы позволяют разделять класс на несколько файлов, а частичные методы — на несколько частей, определённых в разных файлах.

1. system.int32
2. a=2 b=1
3. ****Ошибка:**** Конструкторы не имеют возвращаемого типа, даже **void**. Объявление **public int A() { }** будет воспринято компилятором как объявление обычного метода с именем **A**, возвращающим значение типа **int**.



35





### 1. \*\*`public`\*\* (публичный)

- \*\*Описание\*\*: Доступ разрешен отовсюду, без каких-либо ограничений.

- \*\*Где используется\*\*: Поля, методы, свойства, классы, интерфейсы и конструкторы.

### 2. \*\*`private`\*\* (приватный)

- \*\*Описание\*\*: Доступ разрешен только внутри текущего класса. Поля или методы с этим модификатором недоступны для других классов.

- \*\*Где используется\*\*: Поля, методы, свойства, конструкторы и вложенные классы.

### 3. \*\*`protected`\*\* (защищенный)

- \*\*Описание\*\*: Доступ разрешен внутри текущего класса и классов-наследников. Это позволяет использовать члены класса в дочерних классах, но закрывает доступ для внешнего кода, который не является частью наследования.

- \*\*Где используется\*\*: Поля, методы, свойства, конструкторы.

### 4. \*\*`internal`\*\* (внутренний)

- \*\*Описание\*\*: Доступ разрешен внутри текущей сборки (проекта). Модификатор полезен, если нужно ограничить доступ к членам только внутри одного сборочного файла (DLL или EXE).

- \*\*Где используется\*\*: Поля, методы, свойства, классы и конструкторы.

### 5. \*\*`protected internal`\*\* (защищённый внутренний)

- \*\*Описание\*\*: Доступ разрешен либо из текущей сборки, либо из классов-наследников, даже если они находятся в другой сборке. Это комбинация модификаторов `protected` и `internal`.

- \*\*Где используется\*\*: Поля, методы, свойства, конструкторы.

### 6. \*\*`private protected`\*\* (приватный защищённый)

- \*\*Описание\*\*: Доступ разрешен внутри текущего класса и классов-наследников, но только если они находятся в той же сборке. Это комбинация модификаторов `private` и `protected`, с дополнительным ограничением для доступа из других сборок.

- \*\*Где используется\*\*: Поля, методы, свойства, конструкторы