### 1. Принципы ООП

Инкапсуляция

- Скрытие внутренней реализации объекта и предоставление доступа к данным только через определенные методы. Это защищает данные от некорректного использования.

Наследование

- Возможность создавать новый класс на основе существующего, унаследовав его свойства и методы. Это упрощает код и способствует его повторному использованию.

Полиморфизм

- полиморфизм — это способность обьекта использовать методы производного класса, который не существует на момент создания базового.

Абстракция

Выделение основных характеристик объекта в наружный контекст и сокрытие второстепенных деталей.

### 2. Класс .NET, от которого наследуются все классы

- Все классы в .NET наследуются от класса `System.Object`. Этот класс предоставляет базовые методы, такие как `Equals`, `GetHashCode`, `GetType` и `ToString`.

### 3. Открытые методы System.Object

- `Equals(object obj)` Проверяет, равен ли текущий объект указанному объекту.

- `GetHashCode()` Возвращает хэш-код для текущего объекта.

- `GetType()` Возвращает объект `Type`, представляющий текущий экземпляр.

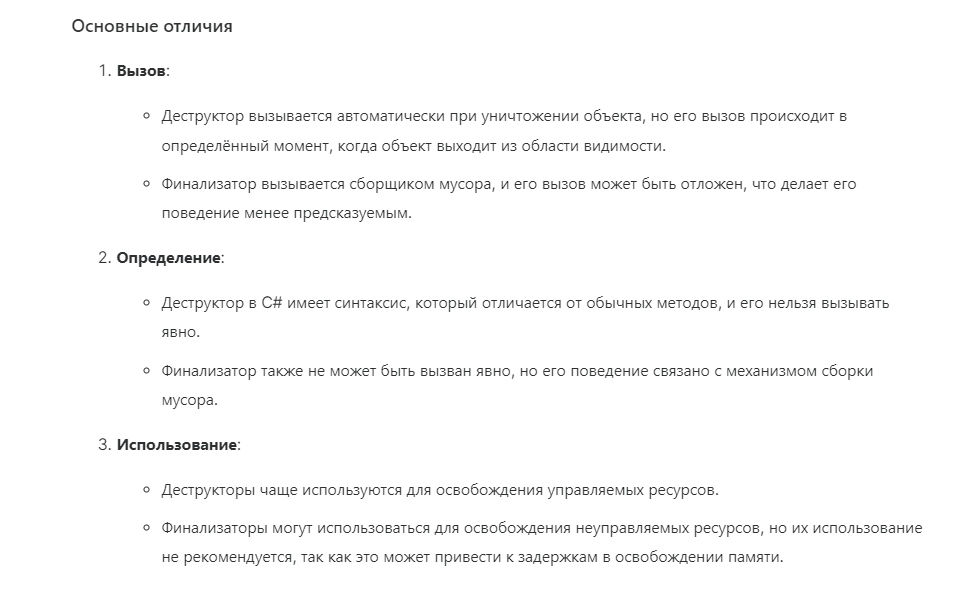
- `ToString()` Возвращает строковое представление текущего объекта.

- `Finalize()` Вызывается сборщиком мусора перед удалением объекта.

### 4. Закрытые методы System.Object

- `Finalize()` Это защищенный метод, который не может быть вызван напрямую. Он используется для освобождения ресурсов перед тем, как сборщик мусора удалит объект.

№№№



### 5. Пример определения класса

```csharp

public class Student

{

public string Name { get; set; }

public int Age { get; set; }

public void Study()

{

Console.WriteLine(${Name} is studying.);

}

}

```

### 6. Ключевые слова, используемые при определении класса

- `public` Доступен из других классов.

- `private` Доступен только внутри класса.

- `protected` Доступен в этом классе и производных от него.

- `internal` Доступен только в пределах одной сборки.

- `sealed` Запрещает наследование от класса.

- `abstract` Используйте модификатор abstract в объявлении класса, чтобы указать, что класс предназначен только для того, чтобы быть базовым классом других классов, а не создавать экземпляры сам по себе

- `static` Определяет класс, который не может быть создан как объект. Вы не можете использовать оператор new для создания экземпляра статического класса.

### 7. Отличие между объектом и классом

- Класс Шаблон или чертеж, который описывает свойства и методы, которые будут у объектов.

- Объект Конкретный экземпляр класса с заданными значениями свойств и способностью выполнять методы, определенные в классе.

### 8. Конструктор и его вызов

- Конструктор Это специальный метод, который вызывается при создании объекта. Он инициализирует объект и задает начальные значения для его свойств.

- Конструктор вызывается автоматически, когда создается новый экземпляр класса.

### 9. Свойства конструктора

- Имя Должно совпадать с именем класса.

- Нет типа возврата Конструкторы не имеют возвращаемого типа.

- Перегрузка Можно создавать несколько конструкторов с различными параметрами.

- Автоматический вызов Вызывается автоматически при создании объекта.

### 10. Что такое деструктор (destructor)

- Деструктор Это специальный метод, который вызывается перед удалением объекта сборщиком мусора. Его задача — освободить ресурсы, занимаемые объектом. В C# деструкторы определяются с помощью тильды (~) перед именем класса.

### 11. Что такое `this`

- `this` Это ссылка на текущий экземпляр класса. Используется для доступа к членам класса (свойствам и методам) из его собственных методов, особенно когда имена параметров и членов совпадают.

Пример использования `this`

```csharp

public class Student

{

private string name;

public Student(string name)

{

this.name = name; Используем this для различия между параметром и полем

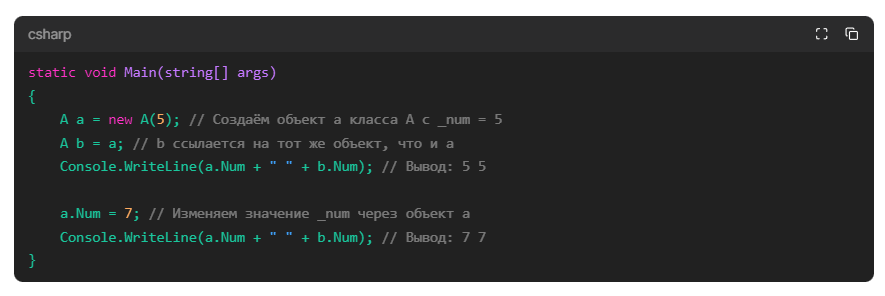
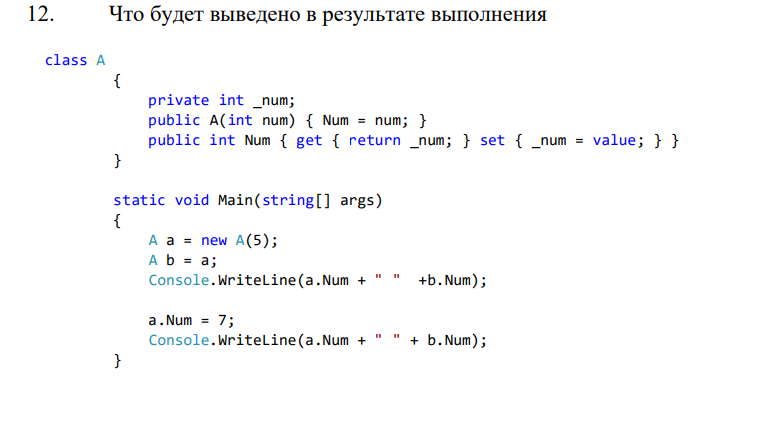
}

}

```

Вот ответы на ваши вопросы по C#:

12.



### 13. Спецификаторы доступа для класса и методов класса

В C# существуют следующие спецификаторы доступа:

- \*\*public\*\*: Доступен из любого кода.

- \*\*private\*\*: Доступен только внутри самого класса.

- \*\*protected\*\*: Доступен внутри класса и его производных.

- \*\*internal\*\*: Доступен только внутри одной сборки.

- \*\*protected internal\*\*: Доступен внутри той же сборки и в производных классах.

### 14. Модификатор `protected internal`

- \*\*`protected internal`\*\*: Позволяет доступ к члену класса из производных классов и из классов в пределах одной сборки. Это комбинация `protected` и `internal`, предоставляющая более гибкий доступ.

### 15. Использование `ref` и `out` параметров функции

- \*\*`ref`\*\*: Позволяет передавать параметры по ссылке, что позволяет изменять значение переменной внутри метода. Переменная, передаваемая с `ref`, должна быть инициализирована перед передачей.

- \*\*`out`\*\*: Также передает параметры по ссылке, но переменная не должна быть инициализирована перед передачей. Обычно используется для возвращения нескольких значений из метода.

Пример:

```csharp

void Example(ref int x, out int y)

{

x += 10; // Изменяет x

y = 20; // Присваивает значение y

}

```

### 16. Пример необязательных и именованных параметров метода

- \*\*Необязательные параметры\*\*: Параметры, которые имеют значения по умолчанию.

```csharp

void Method(int a, int b = 10)

{

Console.WriteLine(a + b);

}

```

- \*\*Именованные параметры\*\*: Позволяют передавать параметры в произвольном порядке, указывая имя параметра.

```csharp

Method(b: 5, a: 3); // Вызов с именованными параметрами

```

### 17. Пример полей класса

```csharp

public class Example

{

public static int StaticField = 5; // Статическое поле

public const double Pi = 3.14; // Константное поле

private readonly int readOnlyField; // Поле только для чтения

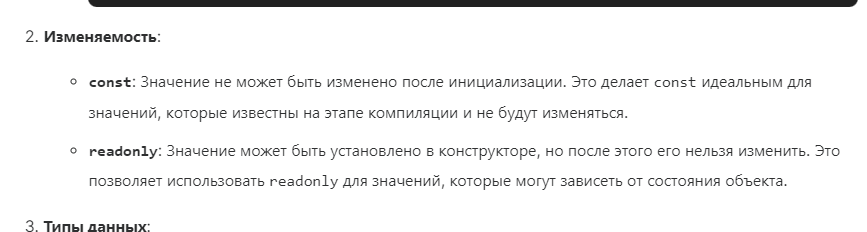
public Example(int value)

{

readOnlyField = value; // Инициализация в конструкторе

}

}



```

### 18. Пример определения свойств класса

```csharp

public class Person

{

private string name;

public string Name

{

get { return name; }

set { name = value; }

}

}

```

\*\*Связь с инкапсуляцией\*\*: Свойства позволяют контролировать доступ к полям класса, обеспечивая защиту данных и возможность добавить логику при получении или установке значений.

### 19. Явное имя параметра для метода `set` свойства класса

- Явное имя параметра для метода `set` — это `value`. Например:

```csharp

public string Name

{

set { name = value; }

}

```

### 20. Автоматические свойства

- \*\*Автоматические свойства\*\* позволяют создавать свойства без явного определения поля. C# автоматически создает скрытое поле для хранения значения.

```csharp

public class Person

{

public string Name { get; set; } // Автоматическое свойство

}

```

### 21. Индексаторы класса

- \*\*Индексаторы\*\* позволяют объектам класса быть индексируемыми, как массивы. Они определяются с помощью ключевого слова `this`.

```csharp

public class SampleCollection

{

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

public int this[int index]

{

get { return arr[index]; }

set { arr[index] = value; }

}

}

```

\*\*Ограничения\*\*: Индексаторы могут принимать только параметры и не могут иметь имя, кроме `this`.

### 22. Перегруженный метод

- \*\*Перегруженный метод\*\* — это метод, который имеет одно и то же имя, но различается по параметрам (тип, количество или порядок).

```csharp

public void Add(int a, int b) { }

public void Add(double a, double b) { }

```

### 23. Частичные классы и их преимущества

- \*\*Частичные классы\*\* позволяют разделять определение класса на несколько файлов. Это удобно для организации кода, особенно в больших проектах.

```csharp

// File1.cs

public partial class MyClass

{

public void Method1() { }

}

// File2.cs

public partial class MyClass

{

public void Method2() { }

}

```

### 24. Анонимный тип в C#

- \*\*Анонимный тип\*\* — это тип, который создается без явного объявления класса. Они используются для временного хранения данных с заданными свойствами.

```csharp

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

```

### 25. Статические классы

- \*\*Статические классы\*\* предназначены для хранения статических методов и свойств. Они не могут создавать экземпляры, что делает их идеальными для хранения утилитарных методов и функций, которые не требуют состояния объекта. и могут содержать только статические члены.

```csharp

public static class MathUtilities

{

public static int Add(int a, int b) => a + b;

}

```

### 26. Отличие статического поля от экземплярного

- \*\*Статическое поле\*\*: Один экземпляр для всех объектов класса. Сохраняет одно и то же значение для всех экземпляров.

- \*\*Экземплярное поле\*\*: У каждого экземпляра класса свое значение. Поле создается для каждого объекта класса.

### 27. Работа статических конструкторов

- \*\*Статические конструкторы\*\* вызываются автоматически при первом доступе к статическим членам класса или при создании первого экземпляра класса. Они используются для инициализации статических полей.

```csharp

public class Example

{

static Example()

{

// Инициализация статических полей

}

}

```

### 28. Поверхностное (shallow) и глубокое (deep) копирование

- \*\*Поверхностное копирование\*\*: Копируются только ссылки на объекты. Если объект содержит ссылки на другие объекты, копия будет ссылаться на те же объекты.

- \*\*Глубокое копирование\*\*: Копируются все данные, включая вложенные объекты. Создаются новые экземпляры для всех объектов.

### 29. Равенство и тождество объектов

- \*\*Равенство\*\*: Проверяет, равны ли значения объектов (сравнение содержимого).

- \*\*Тождество\*\*: Проверяет, ссылаются ли оба объекта на один и тот же экземпляр в памяти.

### 30. Частичные классы и частичные методы

- \*\*Частичные классы\*\*: Позволяют разделять реализацию класса на несколько файлов.

- \*\*Частичные методы\*\*: Методы, которые могут быть частично определены и не требуют реализации в каждом файле.

```csharp

public partial class MyClass

{

partial void OnSomeEvent(); // Объявление частичного метода

}

public partial class MyClass

{

partial void OnSomeEvent()

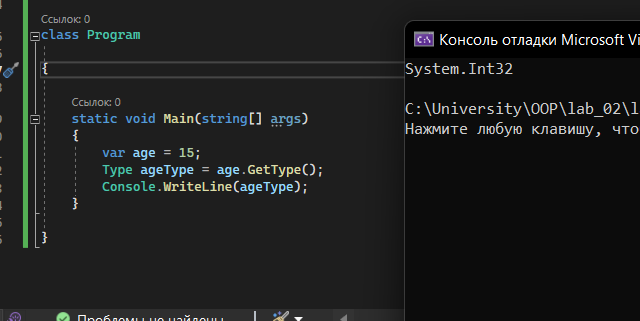
{

// Реализация частичного метода

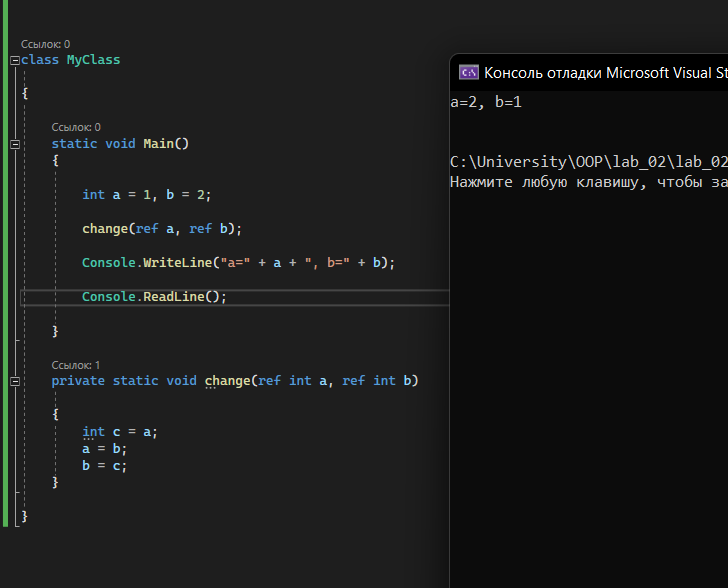
}

}

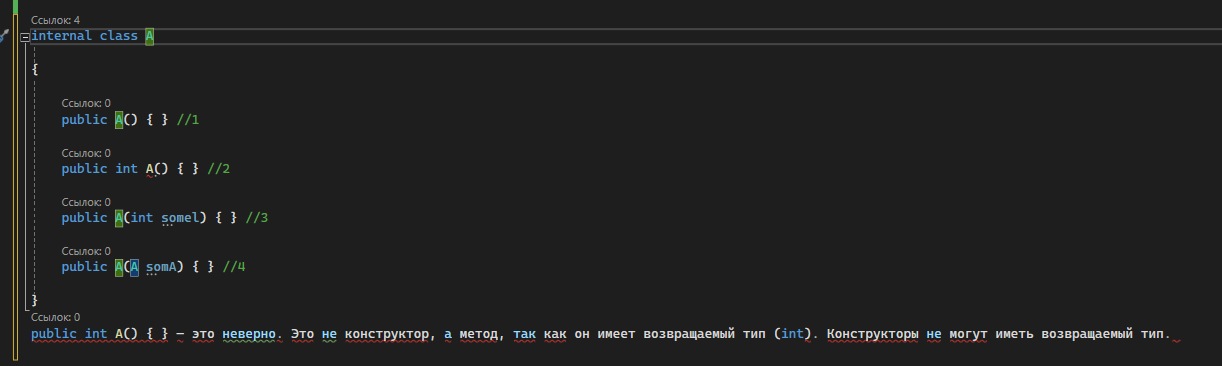
31.



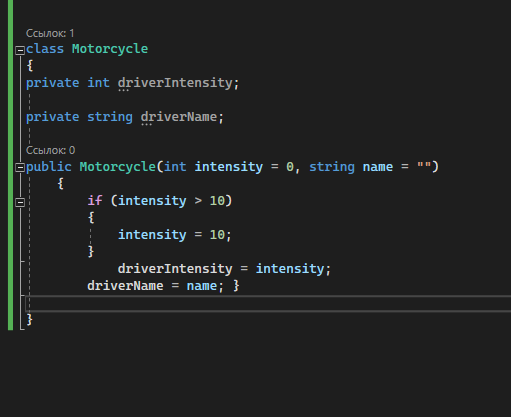
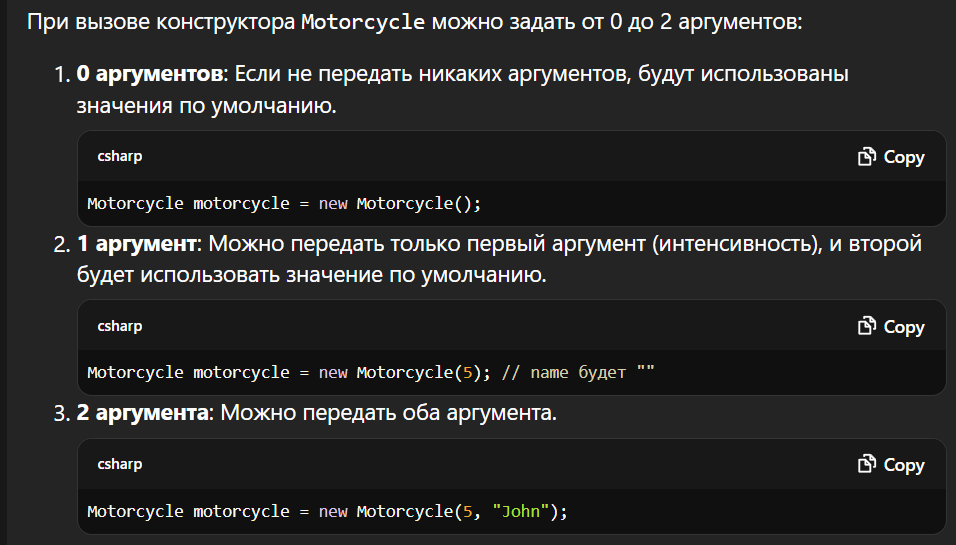
32.



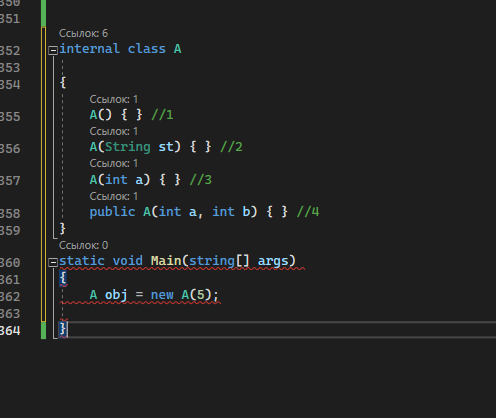
33.

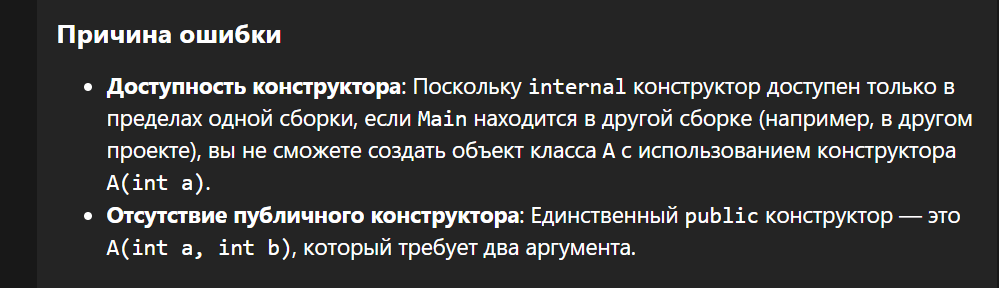


34.

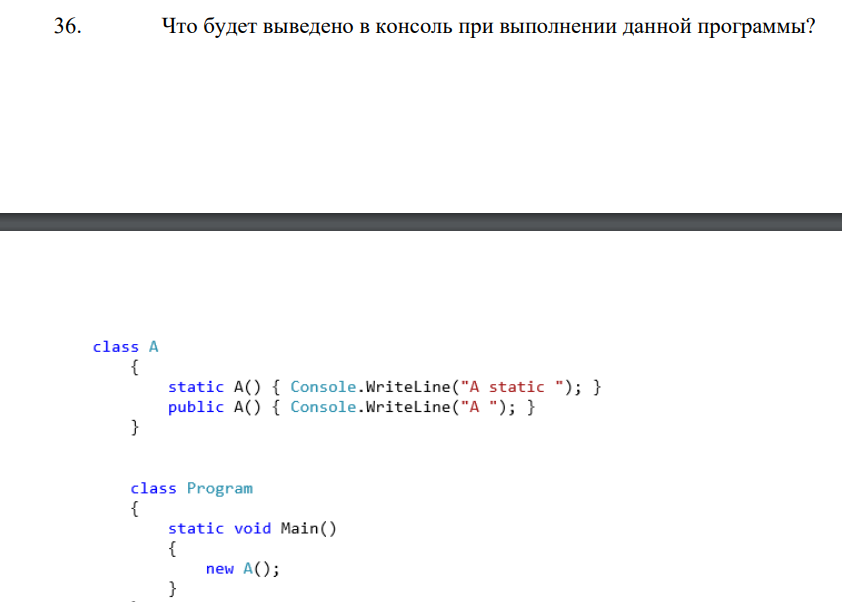


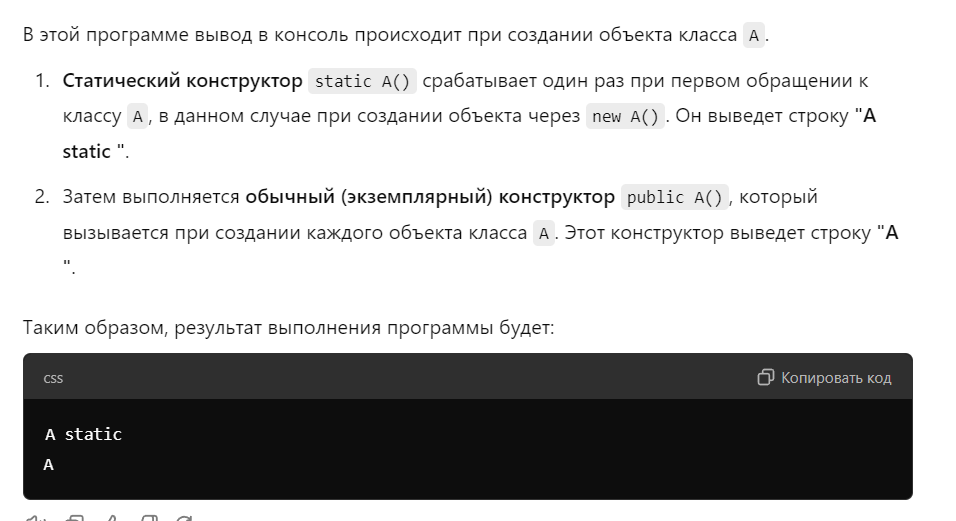
35. Почему не удается создать объект класса A?





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





37. Какая строка приведенного далее класса вызовет ошибку компиляции?

