

1. **Перечислите свойства внутренних и вложенных классов?**

****

**Внутренние классы (Inner Classes)**:

* Определяются внутри другого класса.
* Имеют доступ ко всем членам внешнего класса, включая его приватные поля и методы.
* Могут быть статическими или нестатическими.
* Используются для логического объединения классов, которые имеют смысл только в контексте внешнего класса.

**Вложенные классы (Nested Classes)**:

* Это классы, которые определяются внутри другого класса, но могут быть статическими.
* Вложенные классы не имеют доступа к нестатическим членам внешнего класса без ссылки на экземпляр внешнего класса.
* Используются для упрощения структуры кода и для создания вспомогательных классов, которые используются только в контексте внешнего класса.

**2. Что такое статический класс и какие у него свойства?**

этот класс является набором только статических сущностей (полей, методов, свойств и т. п.), и, внимание, создавать объекты такого класс нельзя!

**Статический класс**:

* Определяется с помощью ключевого слова static.
* Не может быть инстанцирован, то есть нельзя создать его объект.
* Все его члены (методы, свойства, поля) также должны быть статическими.
* Используется для группировки утилитарных методов и функций, которые не зависят от состояния экземпляра класса.
* Например, класс Math в .NET является статическим классом, содержащим методы для математических операций.

**Свойства статического класса**:

 **Не может быть инстанцирован**:

* Вы не можете создать экземпляр статического класса с помощью оператора new. Это означает, что все члены статического класса доступны без создания объекта.

 **Содержит только статические члены**:

* Все члены статического класса должны быть статическими. Это означает, что они принадлежат самому классу, а не его экземплярам.

 **Не может быть унаследован**:

* Статические классы не могут быть унаследованы. Вы не можете создать подкласс для статического класса.

 **Инициализация**:

* Статические классы могут содержать статические конструкторы, которые вызываются автоматически перед первым использованием класса. Статический конструктор используется для инициализации статических членов класса.

 **Доступ к статическим членам**:

* Статические члены могут быть вызваны без создания экземпляра класса, используя только имя класса. Например: Math.Sqrt(4).

 **Область видимости**:

* Статические классы могут иметь различные уровни доступа (public, internal и т.д.), что определяет, кто может их использовать.

**3. Каково назначение перегрузки операторов?**

**Перегрузка операторов** позволяет определять или изменять поведение стандартных операторов (таких как +, -, \*, == и т.д.) для пользовательских классов. Это делает код более читабельным и удобным для работы с классами, позволяя использовать операторы как для встроенных типов данных.

**4. Использование ключевого слова operator**

Ключевое слово operator используется для определения перегруженного оператора в классе. С его помощью можно указать, как будет работать оператор для объектов данного класса. Например:

csharp

Copy

public class Complex

{

public double Real { get; set; }

public double Imaginary { get; set; }

public static Complex operator +(Complex c1, Complex c2)

{

return new Complex { Real = c1.Real + c2.Real, Imaginary = c1.Imaginary + c2.Imaginary };

}

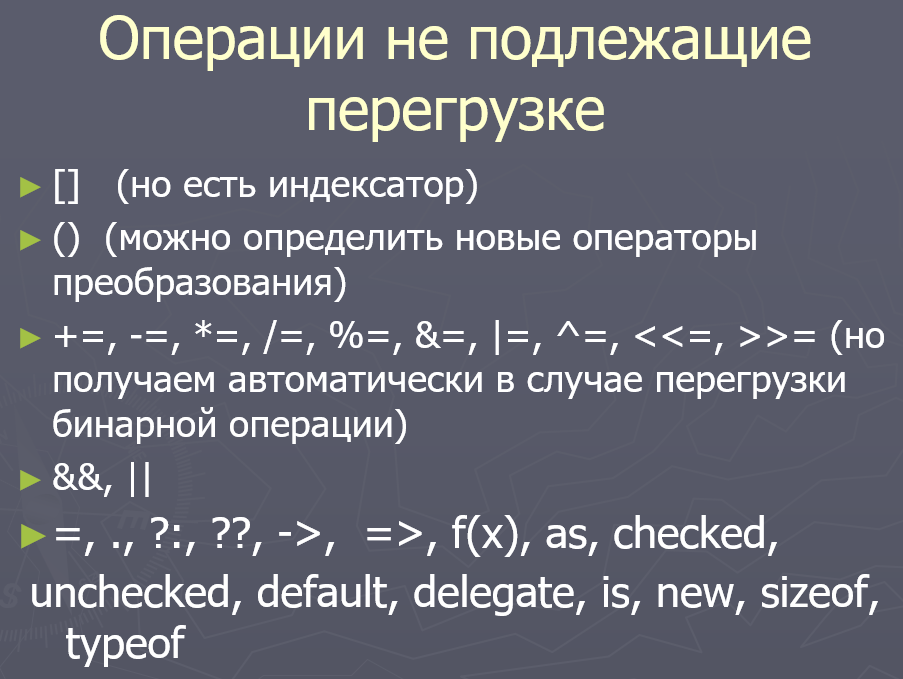
}

В этом примере оператор + перегружен для класса Complex, чтобы складывать два комплексных числа.

**5. Операции, которые нельзя перегружать в C#**

В C# нельзя перегружать следующие операции:

* Операторы :: (разрешение пространства имен).
* Оператор . (доступ к членам).
* Оператор -> (доступ к членам через указатель).
* Оператор new (создание экземпляра объекта).
* Оператор sizeof (определение размера типа).
* Оператор typeof (определение типа).
* Оператор checked и unchecked (управление переполнением).
* Оператор is (проверка типа).
* Оператор as (приведение типов).



**5. Операции, которые нельзя перегружать в C#**

В C# нельзя перегружать следующие операции:

* Оператор :: (разрешение пространства имен).
* Оператор . (доступ к членам).
* Оператор -> (доступ к членам через указатель).
* Оператор new (создание экземпляра объекта).
* Оператор sizeof (определение размера типа).
* Оператор typeof (определение типа).
* Операторы checked и unchecked (управление переполнением).
* Операторы is и as (проверка и приведение типов).

**6. Можно ли перегрузкой отменить очередность выполнения операции?**

Нет, перегрузка операторов не может изменить предшествование или порядок выполнения операций. Правила порядка выполнения остаются фиксированными, и перегрузка операторов не влияет на это. Например, если вы перегрузили оператор +, он будет выполняться в том порядке, который определен правилами выполнения выражений в C#.

**7. Истинно ли следующее утверждение: операция >= может быть перегружена?**

Да, это утверждение истинно. Оператор >= (больше или равно) может быть перегружен для пользовательских классов. Вы можете определить, как этот оператор будет работать с экземплярами вашего класса.

**8. Сколько аргументов требуется для определения перегруженной унарной операции?**

Перегруженная унарная операция требует одного аргумента. Унарные операции, такие как ++, --, +, и -, работают с одним операндом, то есть с одним экземпляром класса:

csharp

Copy

public static MyClass operator ++(MyClass obj)

{

// логика увеличения

return obj;

}

**9. Можно ли перегружать операцию []?**

Да, вы можете перегружать оператор [] для создания индексов в своих классах. Это позволяет вам обращаться к элементам класса так же, как вы обращаетесь к элементам массива или коллекции.

csharp

Copy

public class MyCollection

{

private int[] items = new int[10];

public int this[int index]

{

get { return items[index]; }

set { items[index] = value; }

}

}

**10. Можно ли перегружать операцию ->?**

Нет, оператор -> не может быть перегружен в C#. Этот оператор используется для доступа к членам объекта через указатель и имеет фиксированное поведение, которое нельзя изменить.

### 11. Пример оператора приведения типа

Оператор приведения типа в C# позволяет явно преобразовывать один тип данных в другой. Это может быть сделано с помощью явного или неявного приведения.

**Пример явного приведения**:

csharp

Copy

double number = 9.78;

int intNumber = (int)number; // Явное приведение типа

Console.WriteLine(intNumber); // Вывод: 9

В этом примере переменная number типа double приводится к типу int, что приводит к отбрасыванию дробной части.

**Пример неявного приведения**:

csharp

Copy

int intValue = 123;

long longValue = intValue; // Неявное приведение типа

Console.WriteLine(longValue); // Вывод: 123

В этом случае переменная intValue автоматически приводится к типу long, так как long может хранить все значения int.

### 12. Что такое метод расширения?

**Метод расширения** — это способ добавления новых методов к существующим типам без изменения их исходного кода или создания нового производного типа. Методы расширения являются статическими методами, которые определяются в статическом классе и используют специальный синтаксис this для первого параметра.

#### Как это работает:

1. **Статический класс**: Метод расширения должен находиться в статическом классе.
2. **Статический метод**: Метод должен быть статическим.
3. **Первый параметр**: Первый параметр метода должен иметь модификатор this, который указывает, к какому типу он будет добавлен.

#### Пример метода расширения:

csharp

Copy

public static class StringExtensions

{

public static bool IsNullOrEmpty(this string str)

{

return string.IsNullOrEmpty(str);

}

}

// Использование метода расширения

class Program

{

static void Main()

{

string testString = null;

bool result = testString.IsNullOrEmpty(); // Вызов метода расширения

Console.WriteLine(result); // Вывод: True

}

}

### Как и где использовать методы расширения

* **Удобство**: Они позволяют добавлять методы к существующим классам, не изменяя их исходный код.
* **Соблюдение принципов ООП**: Это позволяет сохранять инкапсуляцию и избегать наследования, когда это не требуется.
* **Читаемость кода**: Методы расширения делают код более читаемым и удобным для использования.

**13. Пусть дан фрагмент кода определения оператора преобразования типа. Определить форму преобразования. public static implicit operator Point2D(Point3D a) {/\* код\*/;}**

В данном фрагменте кода определяется оператор преобразования типа, который позволяет неявно (implicit) преобразовывать объект типа Point3D в объект типа Point2D.

### Форма преобразования

**Неявное (implicit) преобразование** используется в случае, когда преобразование безопасно и не теряет информации. В данном случае это означает, что вы можете безопасно преобразовать Point3D в Point2D, так как Point2D содержит только координаты X и Y, а координата Z в Point3D может быть проигнорирована.

**14. Верные утверждения о методах расширения**

1. **Метод расширения может получать доступ к public членам расширяемого класса.**  
   **Верно.** Методы расширения могут взаимодействовать с общедоступными (public) членами класса.
2. **Метод расширения может получать доступ к protected членам расширяемого класса.**  
   **Неверно.** Методы расширения не могут обращаться к защищенным (protected) членам класса, так как они не являются частью иерархии наследования.
3. **Метод расширения может получать доступ к internal членам расширяемого класса.**  
   **Верно, но с оговоркой.** Методы расширения могут получить доступ к внутренним (internal) членам, если они находятся в одном сборке.
4. **Метод расширения может быть объявлен в любом классе.**  
   **Верно.** Методы расширения могут быть объявлены в любом статическом классе.
5. **Метод расширения может быть без параметров.**  
   **Неверно.** Методы расширения всегда должны иметь хотя бы один параметр, который указывает, к какому типу они применяются (параметр с модификатором this).

**Верные утверждения:**

* 1
* 3 (если в той же сборке)
* 4

**15. Неверное правило перегрузки операторов для C#**

1. **Префиксные операции ++ и – – перегружаются парами.**  
   **Верно.** Если вы перегружаете префиксный ++, вы также должны перегрузить префиксный --.
2. **Операции сравнения перегружаются парами: == и !=; < и >; <= и >=.**  
   **Верно.** Это правило для перегрузки операторов сравнения.
3. **Перегруженные операции обязаны возвращать значения.**  
   **Верно.** Операторы, которые имеют смысл возвращать значение, должны это делать.
4. **Должны объявляться как protected.**  
   **Неверно.** Операторы не обязаны быть защищенными (protected); они могут быть публичными (public) или иметь другие уровни доступа.
5. **True и false можно перегружать.**  
   **Верно.** Вы можете перегружать операторы true и false для пользовательских типов, но это делается через перегрузку операторов true и false.

**Неверное правило:**

* 4