Вот ответы на все 14 вопросов:

1. \*\*Что такое обобщение (generic)?\*\*

Обобщение — это параметризированный тип, позволяющий создавать классы, методы и интерфейсы, которые могут работать с любыми типами данных. Обобщения помогают избежать дублирования кода и повысить его гибкость, позволяя использовать различные типы данных без их явного указания при написании кода.

2. \*\*Пусть дан фрагмент листинга. В какой строчке содержится ошибка?\*\*

Для точного ответа необходимо видеть сам листинг. Чаще всего ошибки в обобщениях возникают из-за несоответствия типов, отсутствия ограничений на параметры обобщения или неправильного их использования.

3. \*\*Как можно наложить определённое ограничение на параметр?\*\*

Для этого используется ключевое слово `where`. Например:

```csharp

class MyClass<T> where T : class

```

В этом примере параметр `T` должен быть ссылочным типом.

4. \*\*Как можно наложить несколько ограничений на параметр?\*\*

Можно указать несколько ограничений через запятую после ключевого слова `where`:

```csharp

class MyClass<T> where T : class, new()

```

Здесь параметр `T` должен быть ссылочным типом и иметь конструктор без параметров.

5. \*\*Перечислите все существующие ограничения на типы данных обобщений:\*\*

- `class`: Тип должен быть ссылочным.

- `struct`: Тип должен быть значимым (например, `int`, `double`).

- Базовый класс: Параметр должен наследовать указанный базовый класс.

- Интерфейс: Параметр должен реализовывать указанный интерфейс.

- `new()`: Параметр должен иметь конструктор без параметров.

6. \*\*Какое ограничение на тип задано в следующем фрагменте листинга?\*\*

```csharp

class A { }

class B : A { }

class C { }

class Test<T> where T : A { }

```

Параметр `T` должен быть либо классом `A`, либо его наследником (как, например, класс `B`).

7. \*\*Какое ограничение на тип задано в следующем фрагменте листинга?\*\*

```csharp

interface A { }

class Test<T> where T : class { }

```

Параметр `T` должен быть ссылочным типом (например, классом).

8. \*\*Какое ограничение на тип задано в следующем фрагменте листинга?\*\*

```csharp

interface A { }

class Test<T> where T : struct { }

```

Параметр `T` должен быть значимым типом (например, `int`, `double`, `bool`).

9. \*\*Примеры, когда обобщённый класс может действовать как базовый или производный класс:\*\*

- Обобщённый класс может быть базовым для других классов:

```csharp

class BaseClass<T> { }

class DerivedClass : BaseClass<int> { }

```

- Также можно наследовать обобщённый класс с обобщённым параметром:

```csharp

class DerivedClass<T> : BaseClass<T> { }

```

10. \*\*В каких случаях в обобщениях может использоваться оператор `default`?\*\*

Оператор `default` используется для получения значения по умолчанию для типа параметра. Он полезен, когда не знаешь, является ли параметр ссылочным или значимым типом. Например:

```csharp

T value = default(T);

```

Если `T` — это ссылочный тип, `value` будет `null`, если значимый — значение по умолчанию (например, 0 для `int`).

11. \*\*Как использовать статические переменные в обобщённых классах?\*\*

Статические переменные в обобщённых классах не общие для всех экземпляров класса. Они уникальны для каждого закрытого типа. Например:

```csharp

class MyClass<T>

{

public static int counter = 0;

}

MyClass<int>.counter = 10;

MyClass<string>.counter = 20;

```

Здесь будет две отдельных переменные `counter` для типов `int` и `string`.

12. \*\*Пример обобщённого интерфейса:\*\*

```csharp

public interface IGenericOperations<T>

{

void Add(T item);

void Remove(T item);

T Find(Predicate<T> match);

}

```

13. \*\*Отличие обобщённых классов от обобщённых структур:\*\*

- Обобщённые классы работают с ссылочными типами, и параметры обобщения могут быть как ссылочными, так и значимыми типами.

- Обобщённые структуры могут быть только значимыми типами, и они не могут быть использованы в качестве базовых классов.

14. \*\*Какие классы для работы с файлами вы знаете? Приведите пример.\*\*

- `StreamReader` и `StreamWriter` для чтения и записи текстовых файлов:

```csharp

using (StreamWriter writer = new StreamWriter("file.txt"))

{

writer.WriteLine("Hello, World!");

}

using (StreamReader reader = new StreamReader("file.txt"))

{

string line = reader.ReadLine();

Console.WriteLine(line);

}

```

- `FileStream`, `BinaryReader`, `BinaryWriter` — для работы с двоичными данными.

- `File` — для простых операций, таких как копирование, удаление и перемещение файлов.

Эти ответы покрывают все вопросы, связанные с обобщениями, их применением и ограничениями.