Обобщение – параметризированный тип, а именно конструкции, которые позволяют писать код, который будет одинаково работать с различными типами данных. Благодаря обобщению, мы можем избежать процесс упаковки и распаковки значимых типов.

Указывая параметр типа, можно наложить определенное ограничение на этот параметр. Это делается с помощью оператора where при указании параметра типа:

**class имя\_класса<параметр\_типа> where параметр\_типа : ограничения { // ...**

где ограничения указываются списком через запятую.

В C# предусмотрен ряд ограничений на типы данных:

**Ограничение на базовый класс**

Требует наличия определенного базового класса в аргументе типа. Это ограничение накладывается указанием имени требуемого базового класса. Разновидностью этого ограничения является неприкрытое ограничение типа, при котором на базовый класс указывает параметр типа, а не конкретный тип. Благодаря этому устанавливается взаимосвязь между двумя параметрами типа.

**Ограничение на интерфейс**

Требует реализации одного или нескольких интерфейсов аргументом типа. Это ограничение накладывается указанием имени требуемого интерфейса.

**Ограничение на конструктор**

Требует предоставить конструктор без параметров в аргументе типа. Это ограничение накладывается с помощью оператора new().

**Ограничение ссылочного типа**

Требует указывать аргумент ссылочного типа с помощью оператора class.

**Ограничение типа значения**

Требует указывать аргумент типа значения с помощью оператора struct.

Среди всех этих ограничений чаще всего применяются ограничения на базовый класс и интерфейс, хотя все они важны в равной степени.

Примеры, когда обобщенный класс может действовать как базовый или производный класс.

class Class1<T>

{

}

// Унаследованный обобщенный класс

class Class2\_1<T> : Class1

{ }

// Ещё один унаследованный класс с собственными параметрами

class Class2\_2<T, V> : Class1

{ }

class Class3<T, V, E, G> : Class2\_2

{ }

// Обычный необобщенный класс

class SomeClass

{ }

// Унаследованный от обычного класса обобщенный класс

class ObClass<T> : SomeClass

{ }

Поскольку, при объявлении обобщенного класса, параметр типа заранее неизвестен, то невозможно определить значение по умолчанию для переменных обобщенного типа. В этом случае применяется ключевое слово default(T).

Подобно классам, структуры также могут быть обобщенными. Они очень похожи на обобщенные классы, за исключением возможности наследования.

Классы для работы с файлами: **File** и **FileInfo.**

**FileInfo**

Некоторые полезные методы и свойства класса **FileInfo**:

* **CopyTo(path)**: копирует файл в новое место по указанному пути path
* **Create()**: создает файл
* **Delete()**: удаляет файл
* **MoveTo(destFileName)**: перемещает файл в новое место
* Свойство **Directory**: получает родительский каталог в виде объекта DirectoryInfo
* Свойство **DirectoryName**: получает полный путь к родительскому каталогу
* Свойство **Exists**: указывает, существует ли файл
* Свойство **Length**: получает размер файла
* Свойство **Extension**: получает расширение файла
* Свойство **Name**: получает имя файла
* Свойство **FullName**: получает полное имя файла

### File

Класс File реализует похожую функциональность с помощью статических методов:

* **Copy()**: копирует файл в новое место
* **Create()**: создает файл
* **Delete()**: удаляет файл
* **Move**: перемещает файл в новое место
* **Exists(file)**: определяет, существует ли файл

Для каждого статического типа мы создаем свое статическое поле.