2. Faker

Необходимо реализовать генератор DTO (объектов для переноса данных) со случайными тестовыми данными:

var faker = new Faker();

Foo foo = faker.Create<Foo>();

Bar bar = faker.Create<Bar>();

При создании объекта следует использовать **конструктор,** а также **публичные поля и свойства с публичными сеттерами**. Следует учитывать сценарии, когда у класса только приватный конструктор, несколько конструкторов, конструктор с параметрами и публичные поля/свойства.

Заполнение должно быть **рекурсивным** (если полем DTO является другой DTO, то он также должен быть создан с помощью Faker). Логика определения, что является DTO, на усмотрение автора.

Реализовать генераторы случайных значений для базовых типов-значений (int, long, double, float, etc), строк, одного любого системного класса, который можно встретить в DTO, на выбор (дата/время, url, etc), коллекций объектов всех перечисленных типов (поддержка разновидностей IEnumerable<T>, List<T>, IList<T>, ICollection<T>, T[] на усмотрение автора, минимум один вариант из приведенных);

Выделить как минимум 2 базовых генератора в отдельные **подключаемые модули** (плагины), которые будут загружаться на старте приложения.

Предусмотреть учет **циклических зависимостей:**

class A

{

public B { get; set; }

}

​

class B

{

public C { get; set; }

}

​

class C

{

public A { get; set; } // циклическая зависимость,

// может быть на любом уровне вложенности

}

Предусмотреть обработку типов, которые не являются DTO, и для которых нет генератора. Их наличие не должно приводить к исключениям во время выполнения.

**Задание со звездочкой**

Настройка генерируемых случайных значений для конкретного поля путем передачи собственного генератора для конкретного поля/свойства:

var config = new FakerConfig();

// настройка может иметь и другой API на усмотрение автора

// (см. ограничение далее по заданию)

config.Add<Foo, string, CityGenerator>(foo => foo.City);

var faker = new Faker(config);

Foo foo = faker.Create<Foo>(); // при заполнении свойства City должен использоваться CityGenerator

**Ограничение:** задавать имя свойства/поля в виде строки (явно или с помощью оператора nameof) запрещено, следует использовать деревья выражений: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/expression-trees/>.

Настройка должна работать и для неизменяемых объектов, свойства которых не имеют публичного доступа для записи, а создание выполняется через конструктор:

public class Person

{

public string Name { get; }

public Person(string name)

{

Name = name;

}

}

Если для такого объекта вы задаете собственный генератор для поля Name config.Add<Person, string, NameGenerator>(p => p.Name), то он должен использоваться при при генерации параметра name для конструктора. При обработке такой ситуации достаточно анализировать имена и типы параметров конструктора, предполагая, что его реализация тривиальна (параметры присваиваются свойствам с соответствующими именами).