Перечисляемый тип (*enumerable type*) – это тип, который имеет экземплярный метод `GetEnumerator()`, возвращающий перечислитель. *Перечислитель* (*enumerator*) – объект, обладающий свойством `Current`, представляющим текущий элемент набора, и методом `MoveNext()` для перемещения к следующему элементу. Оператор `foreach` получает перечислитель, вызвав у объекта метод `GetEnumerator()`, а затем использует `MoveNext()` и `Current` для итерации по набору.

Способы сделать класс перечисляемым:

- Реализовать интерфейсы `IEnumerable` и `IEnumerator`.
- Реализовать универсальные интерфейсы `IEnumerable<T>` и `IEnumerator<T>`.
- Способ, при котором стандартные интерфейсы не применяются

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
public interface IEnumerator
{
    object Current { get; }
    bool MoveNext();
    void Reset();
}
```

Свойство только для чтения `Current` представляет текущий объект. Для обеспечения универсальности данное свойство имеет тип `object`. Метод `MoveNext()` выполняет перемещение на следующую позицию в наборе. Метод возвращает булево значение `true`, если дальнейшее перемещение возможно. Предполагается, что `MoveNext()` нужно вызвать и для получения первого элемента, то есть начальная позиция – «перед первым элементом». Метод `Reset()` сбрасывает позицию в начальное состояние. При записи цикла `foreach` объявляется переменная, тип которой совпадает с типом коллекции.

Итератор (*iterator*) – это блок кода, который порождает упорядоченную последовательность значений. Итератор отличает присутствие одного или нескольких операторов `yield`. Оператор `yield return <выражение>` возвращает следующее значение последовательности, а оператор `yield break` прекращает генерацию последовательности. Итераторы могут использоваться в качестве тела метода, если тип метода – один из интерфейсов `IEnumerator`, `IEnumerator<T>`, `IEnumerable`, `IEnumerable<T>`.

```
public class Shop : IEnumerable<string>
{
    public IEnumerator<string> GetEnumerator()
    {
        foreach (var s in _items)
        {
            yield return s;
        }
    }
}
```

Итераторы реализуют концепцию *отложенных вычислений*. Каждое выполнение оператора `yield return` ведёт к выходу из метода и возврату значения. Но состояние метода, его внутренние переменные и позиция `yield return` запоминаются, чтобы быть восстановленными при следующем вызове.