*Универсальные шаблоны* (*generics*) позволяют при разработке пользовательского типа или метода указать в качестве параметра тип, который конкретизируется при использовании. Универсальные шаблоны применимы к классам, структурам, интерфейсам, делегатам и методам.

Универсальные классы и структуры

Опишем класс Stack как универсальный тип. Для этого используется следующий синтаксис: после имени класса в угловых скобках указывается *параметр типа*. Этот параметр может затем использоваться при описании элементов класса (в нашем примере ‑ методов и массива).

public class Stack<T>

{

private T[] \_items;

public void Push(T item) { . . . }

public T Pop() { . . . }

}

Использовать универсальный тип «как есть» в клиентском коде нельзя, так как он является не типом, а, скорее, «чертежом» типа. Для работы со Stack<T> необходимо объявить и создать *сконструированный тип* (*constructed type*), указав в угловых скобках аргумент типа. Аргумент-тип может быть любым типом. Можно создать любое количество экземпляров сконструированных типов, и каждый из них может использовать разные аргументы типа.

Подчеркнем некоторые особенности сконструированных типов. Во-первых, сконструированный тип не связан отношением наследования с универсальным типом. Во-вторых, даже если классы A и B связаны наследованием, сконструированные типы на их основе этой связи лишены. В-третьих, статические поля, описанные в универсальном типе, уникальны для каждого сконструированного типа.

Ограничения на параметры шаблонов

C# допускает указание *ограничения* (*constraint*) для каждого параметра универсального типа. Только тип, удовлетворяющий ограничениям, может быть применён для записи сконструированного типа.

Ограничения объявляются с использованием ключевого слова where, после которого указывается параметр, двоеточие и список ограничения. Элементом списка ограничения на тип могут являться:

* Ключевое слово class (требование, чтобы тип был ссылочным) или ключевое слово struct (требование, чтобы тип был типом значения).
* Имя класса (требование, чтобы тип приводился к этому классу).
* Интерфейс или список интерфейсов (требование, чтобы тип реализовывал эти интерфейсы).
* Конструкция new() (требование, чтобы у типа был конструктор без параметров).

Порядок элементов в списке ограничений имеет значение. Правильный порядок соответствует порядку в списке, приведенном выше.

Ковариантность и контравариантность

Определим понятия ковариантности и контравариантности для сконструированных типов данных. Для этого введём отношение частичного порядка на множестве ссылочных типов:

Если имеется тип C<T>, а также типы T1 и T2 (T1 ≤ T2), то C<T> назовём:

* *ковариантным*, если C<T1> ≤ C<T2>;
* *контравариантным*, если C<T2> ≤ C<T1>;
* *инвариантным*, если не верно ни первое, ни второе утверждение.

Понятия частичного порядка типов, ковариантности и контравариантности связаны с приведением типов. Тот факт, что тип T1 «меньше» типа T2, означает возможность неявного приведения переменной типа T1 к типу T2. Как указывалось ранее, массивы коварианты (например, массив строк присваивается массиву объектов).

Универсальные классы и структуры инварианты, однако, универсальные интерфейсы могут быть описаны как ковариантные или контравариантные относительно некоего параметра-типа. Чтобы указать на ковариантность относительно параметра T, следует использовать ключевое слово out при описании параметра типа. На контравариантность указывает ключевое слово in при описании параметра типа.

public interface IOutOnly<out T>

{

T this[int index] { get; }

}

public interface IInOnly<in T>

{

void Process(T x);

}

Для обеспечения безопасности типов компилятор отслеживает, чтобы ковариантные параметры всегда использовались как типы возвращаемых значений, а контравариантные параметры являлись типами аргументов. Один универсальный интерфейс может, при необходимости, содержать как ковариантные, так и контравариантные параметры.

Универсальные методы

В некоторых случаях достаточно параметризовать не весь пользовательский тип, а только отдельный метод. *Универсальные методы* (*generic methods*) объявляются с использованием параметров-типов в угловых скобках после имени метода[[1]](#footnote-2). Как и при описании универсальных типов, универсальные методы могут содержать ограничения на параметр-тип.

void PushMultiple<T>(Stack<T> stack, params T[] values)

{

foreach (T value in values)

{… }

}

1. Универсальные методы могут заменить перекрытие методов в пользовательском типе, если алгоритмы работы различных версий перекрытых методов не зависят от типов параметров. [↑](#footnote-ref-2)