# **Generic classes and methods**

Универсальные классы и методы сочетают такие характеристики, как возможность многократного использования, типобезопасность и эффективность, которые не обеспечивают их неуниверсальные аналоги. Универсальные типы наиболее часто используются с коллекциями и методами, которые выполняют с ними операции. Пространство имен ***System.Collections.Generic*** содержит несколько универсальных классов коллекций. Неуниверсальные коллекции, например ***ArrayList***, не рекомендуются и поддерживаются только для обеспечения совместимости.

Обобщенные типы позволяют указать конкретный тип, который будет использоваться. Поэтому определим класс ***Person*** как обощенный:

class Person<T>

{

public T Id { get; set; }

public string Name { get; set; }

public Person(T id, string name)

{

Id = id;

Name = name;

}

}

Угловые скобки в описании ***class Person<T>*** указывают, что класс является обобщенным, а тип ***T***, заключенный в угловые скобки, будет использоваться этим классом. Необязательно использовать именно букву ***T***, это может быть и любая другая буква или набор символов. Причем сейчас на этапе написания кода нам неизвестно, что это будет за тип, это может быть любой тип. Поэтому параметр ***T*** в угловых скобках еще называется универсальным параметром, так как вместо него можно подставить любой тип.

Например, вместо параметра ***T*** можно использовать объект ***int***, то есть число, представляющее номер пользователя. Это также может быть объект ***string***, либо или любой другой класс или структура:

Person<int> tom = new Person<int>(546, "Tom"); // упаковка не нужна

Person<string> bob = new Person<string>("abc123", "Bob");

int tomId = tom.Id; // распаковка не нужна

string bobId = bob.Id; // преобразование типов не нужно

Console.WriteLine(tomId); // 546

Console.WriteLine(bobId); // abc123

Поскольку класс ***Person*** является обобщенным, то при определении переменной после названия типа в угловых скобках необходимо указать тот тип, который будет использоваться вместо универсального параметра ***T***.

В данном случае объекты ***Person*** типизируются типами ***int*** и ***string***:

Person<int> tom = new Person<int>(546, "Tom"); // упаковка не нужна

Person<string> bob = new Person<string>("abc123", "Bob");

Поэтому у первого объекта ***tom*** свойство ***Id*** будет иметь тип ***int***, а у объекта ***bob*** - тип ***string***. И в случае с типом ***int*** упаковки происходить не будет.

При попытке передать для параметра id значение другого типа мы получим ошибку компиляции:

Person<int> tom = new Person<int>("546", "Tom"); // ошибка компиляции

А при получении значения из ***Id*** нам больше не потребуется операция приведения типов и распаковка тоже применяться не будет:

int tomId = tom.Id; // распаковка не нужна

Тем самым мы избежим проблем с типобезопасностью. Таким образом, используя обобщенный вариант класса, мы снижаем время на выполнение и количество потенциальных ошибок.

При этом универсальный параметр также может представлять обобщенный тип:

// класс компании

class Company<P>

{

public P CEO { get; set; } // президент компании

public Company(P ceo)

{

CEO = ceo;

}

}

class Person<T>

{

public T Id { get;}

public string Name { get;}

public Person(T id, string name)

{

Id = id;

Name = name;

}

}

Здесь класс компании определяет свойство CEO, которое хранит президента компании. И мы можем передать для этого свойства значение типа Person, типизированного каким-нибудь типом:

Person<int> tom = new Person<int>(546, "Tom");

Company<Person<int>> microsoft = new Company<Person<int>>(tom);

Console.WriteLine(microsoft.CEO.Id); // 546

Console.WriteLine(microsoft.CEO.Name); // Tom

**Статические поля обобщенных классов**

При типизации обобщенного класса определенным типом будет создаваться свой набор статических членов. Например, в классе ***Person*** определено следующее статическое поле:

class Person<T>

{

public static T? code;

public T Id { get; set; }

public string Name { get; set; }

public Person(T id, string name)

{

Id = id;

Name = name;

}

}

Теперь типизируем класс двумя типами ***int*** и ***string***:

Person<int> tom = new Person<int>(546, "Tom");

Person<int>.code = 1234;

Person<string> bob = new Person<string>("abc", "Bob");

Person<string>.code = "meta";

Console.WriteLine(Person<int>.code); // 1234

Console.WriteLine(Person<string>.code); // meta

В итоге для ***Person<string>*** и для ***Person<int>*** будет создана своя переменная ***code***.

**Использование нескольких универсальных параметров**

Обобщения могут использовать несколько универсальных параметров одновременно, которые могут представлять одинаковые или различные типы:

class Person<T, K>

{

public T Id { get;}

public K Password { get; set; }

public string Name { get;}

public Person(T id, K password, string name)

{

Id = id;

Name = name;

Password = password;

}

}

Здесь класс ***Person*** использует два универсальных параметра: один параметр для идентификатора, другой параметр - для свойства-пароля. Применим данный класс:

Person<int, string> tom = new Person<int, string>(546, "qwerty", "Tom");

Console.WriteLine(tom.Id); // 546

Console.WriteLine(tom.Password);// qwerty

Здесь объект Person типизируется типами ***int*** и ***string***. То есть в качестве универсального параметра ***T*** используется тип ***int***, а для параметра ***K*** - тип ***string***.

**Обобщенные методы**

Кроме обобщенных классов можно также создавать обобщенные методы, которые точно также будут использовать универсальные параметры.

Например:

int x = 7;

int y = 25;

Swap<int>(ref x, ref y); // или так Swap(ref x, ref y);

Console.WriteLine($"x={x} y={y}"); // x=25 y=7

string s1 = "hello";

string s2 = "bye";

Swap<string>(ref s1, ref s2); // или так Swap(ref s1, ref s2);

Console.WriteLine($"s1={s1} s2={s2}"); // s1=bye s2=hello

void Swap<T>(ref T x, ref T y)

{

T temp = x;

x = y;

y = temp;

}

Здесь определен обощенный метод ***Swap***, который принимает параметры по ссылке и меняет их значения. При этом в данном случае не важно, какой тип представляют эти параметры.

При вызове метода ***Swap*** типизируем его определенным типом и передаем ему соответствующие этому типу значения.

## **Общие сведения об универсальных шаблонах**

* Используйте универсальные типы, чтобы получить максимально широкие возможности многократного использования кода, обеспечения безопасности типов и повышения производительности.
* Чаще всего универсальные шаблоны используются для создания классов коллекций.
* Библиотека .NET классов содержит несколько универсальных классов коллекций [System.Collections.Generic](https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.generic) в пространстве имен . Универсальные коллекции следует по возможности использовать вместо классов, таких как [ArrayList](https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.arraylist) в [System.Collections](https://learn.microsoft.com/ru-ru/dotnet/api/system.collections) пространстве имен.
* Вы можете создавать собственные универсальные интерфейсы, классы, методы, события и делегаты.
* Универсальные классы можно ограничить, чтобы они разрешали доступ к методам только для определенных типов данных.
* Сведения о типах, используемых в универсальном типе данных, можно получить во время выполнения с помощью отражения.