Ограничения универсальных типов

С помощью универсальных параметров мы можем типизировать обобщенные классы любым типом. Однако иногда возникает необходимость конкретизировать тип. Например, у нас есть следующий класс Account, который представляет банковский счет:

```
class Account
{
    public int Id { get; private set; } // номер счета
    public int Sum { get; set; }
    public Account(int _id)
    {
        Id = _id;
    }
}
```

Для перевода средств с одного счета на другой мы можем определить класс Transaction, который для выполнения всех операций будет использовать объекты класса Account.

Но у класса Account может быть много наследников: DepositAccount (депозитный счет), DemandAccount (счет до востребования) и т.д. И мы не можем знать, какие именно типы счетов будут использоваться в классе Transaction. Возможно, транзакции будут проводиться только между счетами до востребования. И в этом случае в качестве универсального параметра можно установить тип Account:

Ограничения обобщений

```
class Transaction<T> where T : Account
{
    public T FromAccount { get; set; }  // с какого счета перевод
    public T ToAccount { get; set; }    // на какой счет перевод
    public int Sum { get; set; }        // сумма перевода

    public void Execute()
    {
        if (FromAccount.Sum > Sum)
        {
            FromAccount.Sum -= Sum;
            ToAccount.Sum += Sum;
            Console.WriteLine($"Счет {FromAccount.Id}: {FromAccount.Sum}$ \nСчет {ToAccount.Id}: {ToAccount.Sum}$");
        }
        else
        {
            Console.WriteLine($"Недостаточно денег на счете {FromAccount.Id}");
        }
    }
}
```

С помощью выражения `where T : Account` мы указываем, что используемый тип T обязательно должен быть классом Account или его наследником. Благодаря подобному ограничению мы можем использовать внутри класса Transaction все объекты типа T именно как объекты Account и соответственно обращаться к их свойствам и методам.

Теперь применим класс Transaction в программе:

```
class Program
{
    static void Main(string[] args)
    {
        Account acc1 = new Account(1857) { Sum = 4500 };
        Account acc2 = new Account(3453) { Sum = 5000 };
        Transaction<Account> transaction1 = new Transaction<Account>
        {
            FromAccount = acc1,
            ToAccount = acc2,
            Sum = 6900
        };
        transaction1.Execute();

        Console.ReadLine();
    }
}
```

Следует учитывать, что только один класс может использоваться в качестве ограничения.

В качестве ограничения также может выступать и обобщенный класс:

```
class Program
{
    private static void Main(string[] args)
    {
        Account<int> acc1 = new Account<int>(1857) { Sum = 4500 };
        Account<int> acc2 = new Account<int>(3453) { Sum = 5000 };

        Transaction<Account<int>> transaction1 = new Transaction<Account<int>>
        {
            FromAccount = acc1,
            ToAccount = acc2,
            Sum = 6900
```

Ограничения обобщений

```
            };
            transaction1.Execute();

        Console.Read();
    }
}
class Account<T>
{
    public T Id { get; private set; } // номер счета
    public int Sum { get; set; }
    public Account(T _id)
    {
        Id = _id;
    }
}
class Transaction<T> where T : Account<int>
{
    public T FromAccount { get; set; }  // с какого счета перевод
    public T ToAccount { get; set; }    // на какой счет перевод
    public int Sum { get; set; }        // сумма перевода

    public void Execute()
    {
        if (FromAccount.Sum > Sum)
        {
            FromAccount.Sum -= Sum;
            ToAccount.Sum += Sum;
            Console.WriteLine($"Счет {FromAccount.Id}: {FromAccount.Sum}$ \nСчет {ToAccount.Id}: {ToAccount.Sum}$");
        }
        else
        {
            Console.WriteLine($"Недостаточно денег на счете {FromAccount.Id}");
        }
    }
}
```

Ограничения обобщений

В данном случае класс Transaction типизирован классом Account<int>. Класс Account же может быть типизирован абсолютно любым типом. Однако класс Transaction может использовать только объекты класса Account<int> или его наследников. То есть следующий код ошибочен и работать не будет:

```
Account<string> acc1 = new Account<string>("34") { Sum = 4500 };
Account<string> acc2 = new Account<string>("45") { Sum = 5000 };

// так нельзя написать, так как Bank должен быть типизирован классом Account<int> или его наследником
Transaction<Account<string>> transaction1 = new Transaction<Account<string>>
{
    FromAccount = acc1,
    ToAccount = acc2,
    Sum = 900
};
```

В качестве ограничений мы можем использовать следующие типы:

- Классы
- Интерфейсы
- class - универсальный параметр должен представлять класс
- struct - универсальный параметр должен представлять структуру
- new() - универсальный параметр должен представлять тип, который имеет общедоступный (public) конструктор без параметров

Ограничения обобщений

Стандартные ограничения

Есть ряд стандартных ограничений, которые мы можем использовать. В частности, можно указать ограничение, чтобы использовались только структуры или другие типы значений:

```
class Account<T> where T : struct
{ }
```

При этом использовать в качестве ограничения конкретные структуры в отличие от классов нельзя.

Также можно задать в качестве ограничения ссылочные типы:

```
class Transaction<T> where T : class
{ }
```

А также можно задать с помощью слова **new** в качестве ограничения класс или структуру, которые имеют общедоступный конструктор без параметров:

```
class Transaction<T> where T : new()
{ }
```

Если для универсального параметра задано несколько ограничений, то они должны идти в определенном порядке:

1. Название класса, `class`, `struct`. Причем мы можем одновременно определить только одно из этих ограничений
2. Название интерфейса
3. `new()`

```
interface IAccount
{
    int CurrentSum { get; set; }
}
class Person
{
    public string Name { get; set; }
}
class Transaction<T> where T : Person, IAccount, new(){ }
```

Использование нескольких универсальных параметров

Если класс использует несколько универсальных параметров, то последовательно можно задать ограничения к каждому из них:

```
class Transaction<U, V>
        where U : Account<int>
        where V : struct
{

}
```

Ограничения методов

Подобным образом можно использовать и ограничения методов:

```
private static void Main(string[] args)
{
    Account<int> acc1 = new Account<int>(1857) { Sum = 4500 };
    Account<int> acc2 = new Account<int>(3453) { Sum = 5000 };

    Transact<Account<int>>(acc1, acc2, 900);

    Console.Read();
}

public static void Transact<T>(T acc1, T acc2, int sum) where T : Account<int>
{
    if (acc1.Sum > sum)
    {
        acc1.Sum -= sum;
        acc2.Sum += sum;
    }
    Console.WriteLine($"acc1: {acc1.Sum}   acc2: {acc2.Sum}");
}
```

Метод Transact в качестве ограничения принимает тип Account<int>.