ЛЕКЦИЯ 10

- Модуль 3
- 07.02.2024
- Основы понятия вариантности
- Обобщённые методы и делегаты, наследование обобщённых типов

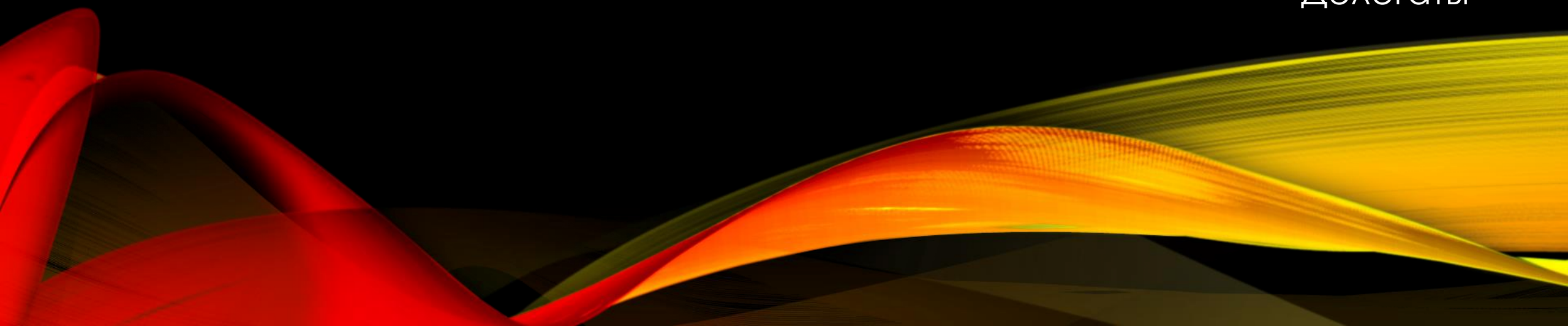
ЦЕЛИ ЛЕКЦИИ

- Познакомится с понятием вариантности
- Познакомится с обобщёнными методами
- Изучить обобщённые делегаты-типы
- Разобрать особенности реализации наследования обобщённых типов

КОВАРИАНТНОСТЬ И КОНТРАВАРИАНТНОСТЬ

Массивы

Делегаты



ОПРЕДЕЛЕНИЯ

- Вариантность – свойство преобразования типов операторами
- Оператор – любая сущность языка C#, преобразующая тип данных в производные от него
  - Определение вариантности в C# (https://habr.com/ru/sandbox/91751/), в статье есть неточности в примерах, будьте внимательны

Преобразование затрагивает один тип

```
T[]

Action <T>
```

Преобразование затрагивает два типа

```
T1 MethodId(T2)

Func<T1, T2>
```

СООТНЕСЕНИЕ ТИПОВ

- T > U «тип T больше типа U»
  - Экземпляр типа T можно заменить экземпляром типа U
- T < U «тип T меньше типа U»
  - Экземпляр типа U можно заменить экземпляром типа T
- T = U «тип T равен типу U»
  - Замены U на T и T на U равновозможны
- T U «тип T не сравним с типом U»
  - Замены не возможны, типы не сравнимы

Здесь оператор «больше», «меньше» и «равно» не понимается в арифметическом смысле

СВОЙСТВА ПРЕОБРАЗОВАНИЯ

Преобразование, осуществляемое оператором, проявляет свойство:

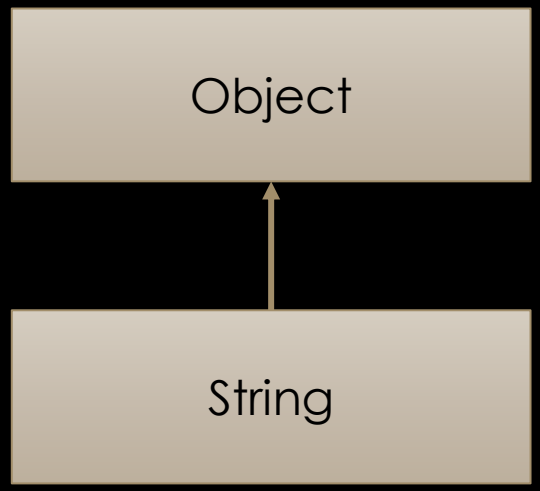
- Ковариантности, если оно сохраняет отношение между парой типов после их преобразования в производные
- Контравариантности, если оно заменяет отношение «больше» на «меньше», но сохраняет «равны» и «не сравнимы»

СВОЙСТВА ПРЕОБРАЗОВАНИЯ. ПРИМЕР (1)

Ковариантность сохраняет отношение между парой типов после их преобразования в производные, т.е. позволяет использовать более конкретный тип

```
Object[] arrObject = new Object[10];
arrObject = new String[2] { "abc", "kbk"};
```

Оператор T[] **ковариантно** преобразует T, допустима замена объекта Object[] на String[]



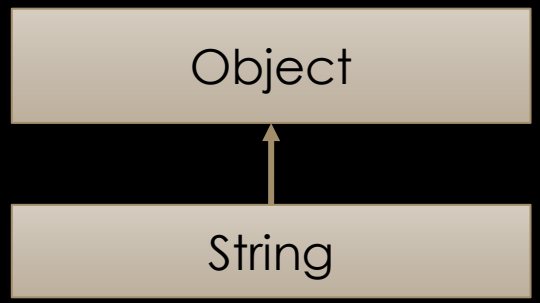
Object > String

СВОЙСТВА ПРЕОБРАЗОВАНИЯ. ПРИМЕР (2)

Контравариантность заменяет отношение «больше» на «меньше», но сохраняет «равны» и «не сравнимы», т.е. позволяет использовать более общий тип

Action<T> **контравариантно** преобразует T, допустима замена **объекта** Action<string> на Action<object>

Object > String




```
public class A {
    public void ConvertString(string str)
    {
        str.ToUpper();
    }
    public void ConvertObject(object str) {  }
}
```

```
A a = new A();
Action<string> ex = a.ConvertString;
Action<string> ex1 = a.ConvertObject;
```

Замена происходит объекта, замена типа ссылки с Action<string> на Action<object> приведёт к ошибке компиляции

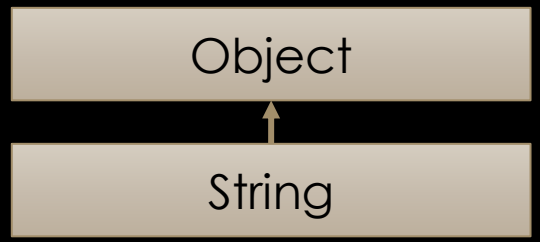
СВОЙСТВА ПРЕОБРАЗОВАНИЯ. ПРИМЕР (3)

Ковариантность сохраняет отношение между парой типов после их преобразования в производные

Контравариантность заменяет отношение «больше» на «меньше», но сохраняет «равны» и «не сравнимы»

Fuct<T, TRes> **ковариантно** преобразует TRes и **контравариантно** преобразует T, допустима замена **объекта** Func<string, Object> на Func<Object, string>

Object > String




```
public class A {
    public string ConvertString(object str) =>
                        "aaa";
    public object ConvertObject(string str) =>
            str;
 }
```

```
A a = new A();
Func<string, Object> ex = a.ConvertString;
Func<string, Object> ex1 = a.ConvertObject;
//Func<Object,string> ex2 = a.ConvertObject;
так нельзя
```

КОВАРИАНТНОСТЬ И КОНТРАВАРИАНТНОСТЬ В C#

- Свойства вариантности представляют собой способ переноса наследования типов на производные от них типы — контейнеры, обобщённые типы, делегаты и т. п.
- С помощью ковариантности и контравариантности можно неявно преобразовывать ссылки на типы коллекций, типы делегатов и аргументы обобщений

Ковариантность сохраняет совместимость операции присваивания, а контравариантность заменяет ее на обратную

Оператор = **ковариантно** преобразует T, допустима замена объекта Object на String

```
// напоминание - совместимость типов при операции присваивания:
string str = "line";
// Приведение производного типа к базовому:
object obj = str;
```

КОВАРИАНТНОСТЬ

Ковариантность позволяет использовать производный тип с большей глубиной наследования, чем задано изначально

Ковариантность «сохраняет иерархию наследования»

```
// Ссылки в C# ковариантны: по ссылке типа родителя
// всегда можно разместить объект типа наследника:
Base derived = new Derived();

public class Base { }
public class Derived : Base { }
```

КОНТРАВАРИАНТНОСТЬ

Контравариантность позволяет использовать более общий тип (с меньшей глубиной наследования), чем заданный изначально

```
class Base { }
class Derived : Base { }
```

Контравариантность «обращает иерархию наследования»

- Предполагается приведение объекта базового типа к производному типу:
  - `Derived d = Base b;`

ПРИМЕР

```
public struct Department
{
    public string Name { get; set; }
    public override string ToString() => $"Department={Name}";
}
```

```
public class Human
{
    public string Name { get; set; }
    public int Age { get; set; }
    public Human() { }
    public Human(string name, int age) => (Name, Age) = (name, age);
    public override string ToString() => $"Name={Name},Age={Age}";
}
```

```
public class Professor : Human
{
    public Department department { get; set; }
    public Professor() { }
    public Professor(string name, int age, Department department) : base(name, age) =>
        this.department = department;
    public override string ToString() => $"{base.ToString()}, {department}";
}
```

ПРИМЕР

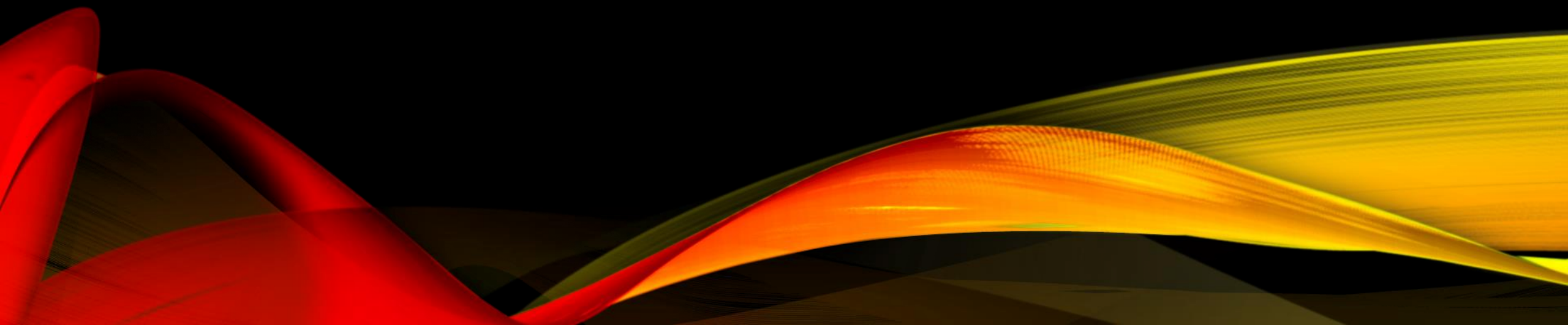
```
// Для примера берём делегат-тип с возвратом базового типа
// и ссылкой на массив также базового типа.
public delegate Human DealWithProfs(Human[] profs);
```

```
public class Program
{
    // Тип возрата – наследник Human.
    public static Professor GetOne(Human[] humans) => new Professor();
    public static Human GetTwo(Professor[] profs) =>  new Human();

    public static void Main()
    {
        Human[] hums = { new Human(), new Human(), new Human()};
        Professor[] profs = {new Professor(), new Professor()};
        // Нет точного совпадения с сигнатурой делегата по типу возвращаемого значения.
        // Преобразование возможно за счёт ковариантности (тип возврата шире).
        DealWithProfs dp = GetOne;
        Professor p = GetOne(hums); // С вызовом всё ок!
        Professor p1 = GetOne(profs); // С вызовом всё ок – проявляется ковариантность массива.

        Human[] hums2 = profs; // Явное проявление ковариантности массива.
        // Так нельзя. Тип параметра шире типа параметра делегата.
        // DealWithProfs dp1 = GetTwo;
    }
}
```

ОБОБЩЁННЫЕ МЕТОДЫ



ОБОБЩЁННЫЕ МЕТОДЫ

Обобщённые методы могут быть объявлены в:

- Классах
- Структурах
- Интерфейсах

Все остальные функциональные члены типов в C# обобщены быть НЕ могут.

Не стоит путать использование параметров типа, объявленных в обобщённом типе, методом и обобщённые методы

СИНТАКСИС ОБОБЩЁННОГО МЕТОДА

Имя обобщённого класса

Описание метода

```
class ClassIdentifier {
{
  …
  return_type MethodIdentifier<T1, T2, …, TN> (param_list) { … }
  …
}
```

Имена обобщённых типов, используемых в классе

Ссылка на экземпляр класса, которому принадлежит метод

Вызов метода

```
obj.MethodIdentifier(parameters)
obj.MethodIdentifier<T1,…,TN>(parameters)
```

Названия конкретных типов, выступающих параметрами

ПРИМЕР ОБОБЩЁННОГО МЕТОДА

```
public static class SwapUtil  {
    public static void Swap<T>(ref T lhs, ref T rhs)  {
        T temp = lhs;
        lhs = rhs;
        rhs = temp;
    }
}
```

Вывод:

```
leftInt: 10, rightInt: 20
leftStr: right, rightStr: left
```

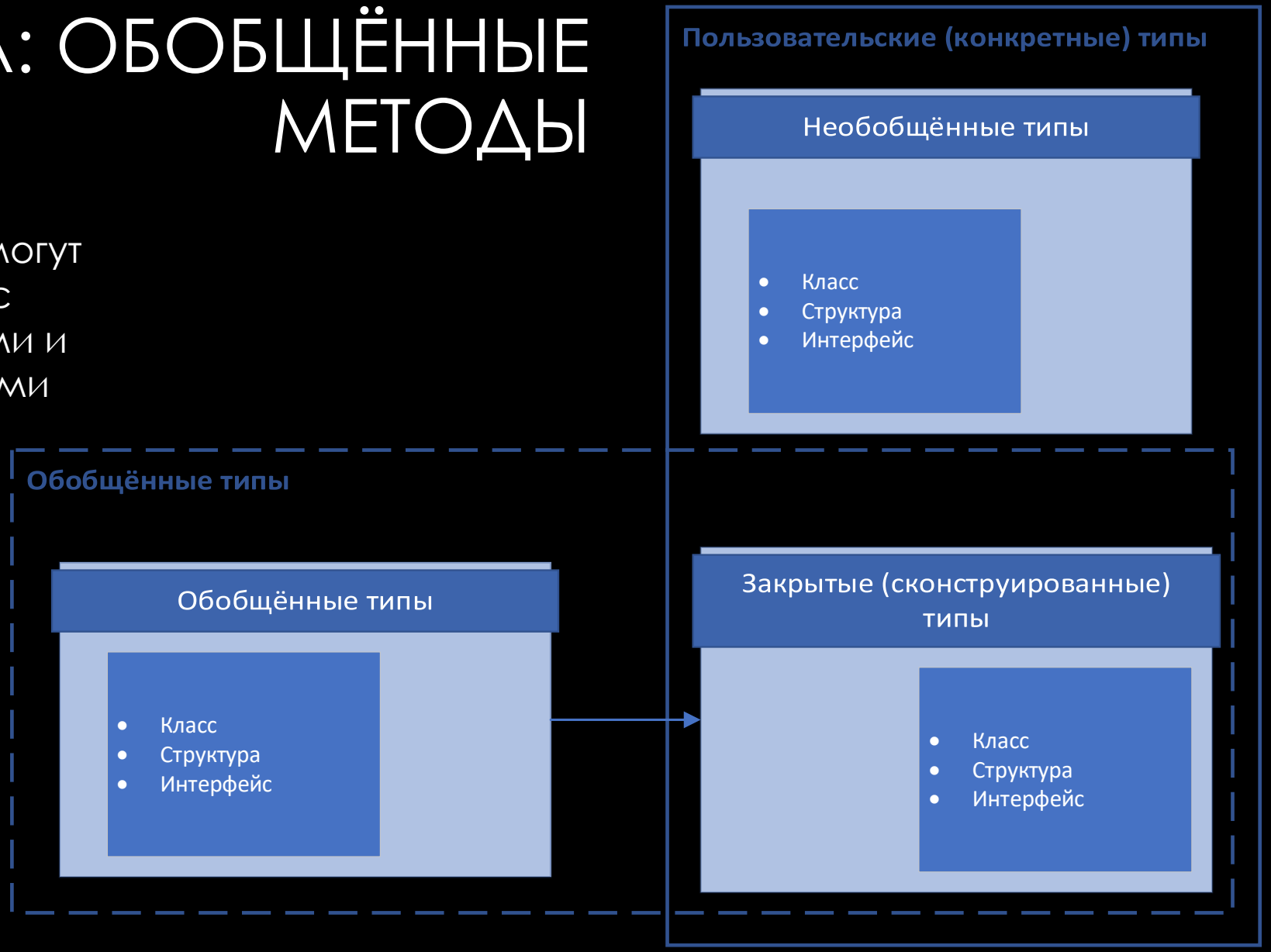
```
int leftInt = 20, rightInt = 10;
string leftStr = "left", rightStr = "right";
SwapUtil.Swap<int>(ref leftInt, ref rightInt);
SwapUtil.Swap(ref leftStr, ref rightStr);
System.Console.WriteLine($"leftInt: {leftInt}, rightInt: {rightInt}");
System.Console.WriteLine($"leftStr: {leftStr}, rightStr: {rightStr}");
```

При подстановке не обязательно указывать типы: компилятор способен вывести их из аргументов.

Исходный код (https://replit.com/@olgamaksimenkova/GenericSwapMethod)

СХЕМА: ОБОБЩЁННЫЕ МЕТОДЫ

Обобщенные методы могут содержать объявления с необобщёнными типами и открытыми обобщёнными типами



РАЗЛИЧНЫЕ АРГУМЕНТЫ ОБОБЩЁННОГО МЕТОДА

```
void MethodName<T1, T2>(T1 t1, T2 t2)
{
    T1 var1 = t1;
    T2 var2 = t2;
    ...
}
```

```
...
MethodName<short, int> (sVal, iVal);
...
```

```
...
MethodName<int, long> (iVal, lVal);
...
```

```
void MethodName<short, int>
(short t1, int t2)
{
    short var1 = t1;
    int var2 = t2;
    ...
}
```

```
void MethodName<int, long>
(int t1, long t2)
{
    int var1 = t1;
    long var2 = t2;
    ...
}
```

ПРИМЕР ОБОБЩЁННЫЙ МЕТОД

```
public static class ArrayReverseTool   {
    static public void ReverseAndPrint<T>(T[] arr)   {
        Array.Reverse(arr);
        Array.ForEach(arr, elem => Console.Write(elem + " "));
        Console.WriteLine();
    }
}
```

Вывод:

```
11 9 7 5 3
3 5 7 9 11
third second first
first second third
2.345 7.891 3.567
3.567 7.891 2.345
```

```
int[] intArray = { 3, 5, 7, 9, 11 };
string[] stringArray = { "first", "second", "third" };
double[] doubleArray = { 3.567, 7.891, 2.345 };
ArrayReverseTool.ReverseAndPrint<int>(intArray);
ArrayReverseTool.ReverseAndPrint(intArray);
ArrayReverseTool.ReverseAndPrint<string>(stringArray);
ArrayReverseTool.ReverseAndPrint(stringArray);
ArrayReverseTool.ReverseAndPrint<double>(doubleArray);
ArrayReverseTool.ReverseAndPrint(doubleArray);
```

Исходный код (https://replit.com/@olgamaksimenkova/DemoGenericMethods4Array)

ПРИМЕР ОБОБЩЁННОГО МЕТОДА РАСШИРЕНИЯ

```
Holder<int> intHolder = new(3, 5, 7);
Holder<string> stringHolder = new("a1", "b2", "c3");
intHolder.Print();
stringHolder.Print();
```

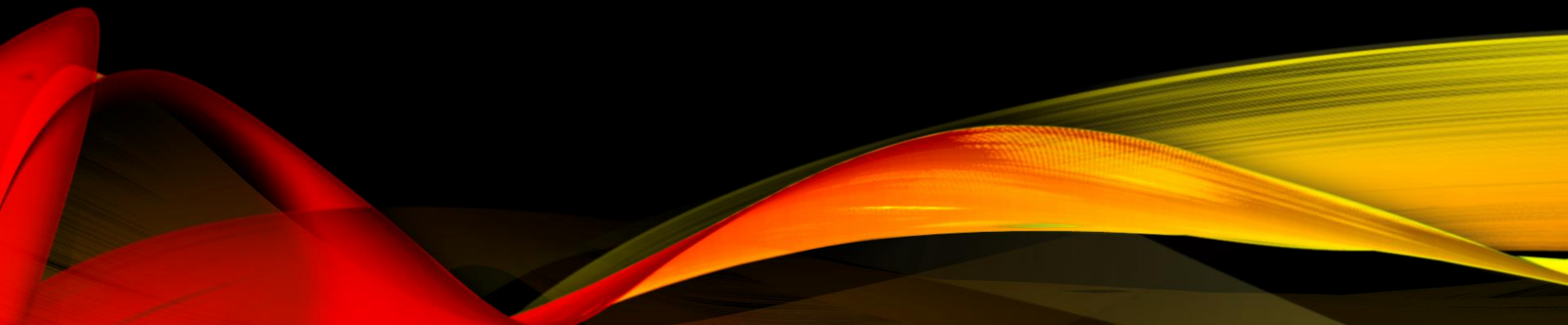
Вывод:

```
3,      5,      7
a1,     b2,     c3
```

```
public static class ExtendHolder   {
    public static void Print<T>(this Holder<T> h) {
        T[] vals = h.Values;
        System.Console.WriteLine($"{vals[0]},\t{vals[1]},\t{vals[2]}");
    }
}
public class Holder<T>
{
    public T[] Values { get; init; } = new T[3];
    public Holder(T v0, T v1, T v2)
        => (Values[0], Values[1], Values[2]) = (v0, v1, v2);
}
```

Исходный код (https://replit.com/@olgamaksimenkova/GenericExtMethod)

ОБОБЩЁННЫЙ ДЕЛЕГАТ



СИНТАКСИС ОБОБЩЁННОГО ДЕЛЕГАТА

Объявление делегата

Имя обобщённого делегата

```
delegate ret_type DelegateIdentifier <T1, T2, …., TN> (arg_list);
```

Имена обобщённых типов, которые используются делегатом

Список аргументов, которые получает делегат

Ссылка с типом обобщённого делегата

Имя обобщённого делегата-типа

Ссылка на обобщенный делегат

```
DelegateIdentifier<T1, T2,…, TN> refDel<t1, t2, …, tN>;
```

Названия конкретных типов, выступающих параметрами

ОБОБЩЁННЫЕ ДЕЛЕГАТ-ТИПЫ

Делегат-типы могут быть обобщёнными:

```
public delegate R GenericDelegate1<T, R>(T value);
public delegate TR Func<T1, T2, TR>(T1 p1, T2 p2);
public delegate void RefAction<T1, T2>(ref T1 p1, ref T2 p2);
```

- Необходимость в создании собственных обобщённых делегатов-типов практически отсутствует, т. к. в библиотеке определены стандартные обобщённые делегаты Action и Func

Практическая необходимость в определении дополнительных обобщённых делегат-типов возникает исключительно в сценарии, когда параметры типов должны использоваться с модификаторами ref, in или out

ПРИМЕР: ОБОБЩЁННЫЙ ДЕЛЕГАТ-ТИП

```
using System;

var myDel = new Func<int, int, string>(PrintString);
Console.WriteLine($"Total: {myDel(15, 13)}");

static string PrintString(int p1, int p2) => (p1 + p2).ToString();

public delegate TR Func<T1, T2, TR>(T1 p1, T2 p2);
```

Обобщённый делегат-тип.

Вывод:

```
Total: 28
```

ИСПОЛЬЗОВАННАЯ И РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

- https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/generics/constraints-on-type-parameters
- https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/generics/constraints-on-type-parameters
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/generic-methods
- https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/covariance-contravariance/
- https://ru.wikipedia.org/wiki/Ковариантность_и_контравариантность_(программирование)