interface ISomeInterface

{

void Call();

}

struct SomeStruct : ISomeInterface

{

public void Call()

{ }

}

class SomeClass

{

public void Run()

{

var someStruct = new SomeStruct();

SomeMethod(someStruct);

}

public void SomeMethod(ISomeInterface @interface)

{

@interface.Call();

}

}

struct SomeStruct : IDisposable

{

void IDisposable.Dispose() {}

}

class SomeClass

{

public void Run()

{

var some = new SomeStruct();

//...

((IDisposable)SomeStruct).Dispose();

}

public void Obects()

{

//В случаях, когда тип переменной выводится неявно из типа другой переменной, бывают случаи, когда нас не устраивает автоматически выведенный тип. Пример:

var list = new[] { 1, 2 }.ToList();

list.Add("ой");

//Мы хотим получить список object'ов, но выведение типов даёт нам список int'ов. Мы можем написать

var list = new[] { (object)1, 2 }.ToList();

list.Add("ой");

так всё будет компилироваться.

//Ещё один тесно связанный случай — тернарный оператор. Если типы альтернатив различны, компилятор не может найти общий тип выражения, и приходится помогать:

Animal animal = nya ? new Cat() : new Dog(); // не компилируется в C# <9

Animal animal = nya ? (Animal)new Cat() : new Dog(); // компилируется

//(Очень похожая проблема возникает с Nullable-типами: int? result = good ? 1 : null требует явного преобразования одного из операндов-альтернатив.)

//Этот случай подсказал @Pavel Mayorov в комментариях, спасибо! Впрочем, как указывает @EvgeniyZ, в C# 9 с target typing в некоторых случаях преобразование типов не нужно:

Animal animal = nya ? new Cat() : new Dog(); // компилируется в C#9

var animal = nya ? new Cat() : new Dog(); // не компилируется в C#9

var animal = nya ? (Animal)new Cat() : new Dog(); // компилируется

}

static public IEnumerable<R> MultiZip<T, R>(

this IEnumerable<List<T>> sequences,

Func<IEnumerable<T>, R> resultSelector)

{

var enumerators =

sequences.Select(s => (IEnumerator<T>)s.GetEnumerator()).ToList();

try

{

while (enumerators.All(en => en.MoveNext()))

yield return resultSelector(enumerators.Select(en => en.Current));

}

finally

{

foreach (var en in enumerators)

en.Dispose();

}

}

}