ПИШНИК: BECOME HUMAN

Глава 13: Интерфейсы

Автор презентации – Сагалов Даниил БПИ-196

Интерфейсы в C# 7.3 и Ранее

```
public interface IKycachiy
{
    Ссылок: 0
    void Kycb(Object target);
    Ссылок: 0
    void Tbmok(Object target);
}
```

Интерфейсы – ссылочные типы данных, содержащие определение какого-либо функционала, который обязаны реализовать классы или структуры. Для определения интерфейса используйте ключевое слово **interface**. По соглашению об именах интерфейсы начинаются с буквы **I**. Вы не можете создавать объекты типов интерфейсов напрямую.

Хотя механизм работы с интерфейсами схож с наследованием абстрактных классов, принято говорить, что класс реализует данный интерфейс (а не наследует). Класс может реализовывать **более одного** интерфейса, если указать их после двоеточия как при наследовании через запятую. При наследовании список интерфейсов должен идти **после класса-родителя**, в противном случае возникнет ошибка компиляции.

Интерфейсы могут содержать в себе только функциональные члены класса, а именно: **методы**, **свойства**, **события** (кроме настраиваемых, до C# 8.0) и **индексаторы**, причём все эти члены не должны определять тело (до C# 8.0). Помните, что вы не можете указать модификатор доступа членов интерфейса – все члены открытые по умолчанию (неактуально для C# 8.0). Кроме того, по своей природе интерфейсы не могут быть объявлены как **static** или **sealed**, в них никогда не бывает конструкторов (статические – с C# 8.0) или деструкторов. Модификаторы членов **virtual** и **override** тоже недоступны, так как члены интерфейса не могут иметь реализации. Интерфейсы могут наследовать только другие интерфейсы.

Интерфейсы в C# 8.0

Начиная с версии C# 8.0, интерфейсы получили достаточно большое количество нового функционала. Основные нововведения – возможность добавлять статические поля и константы, использовать модификаторы доступа, а также определять реализацию членов по умолчанию.

Дополнительно теперь вы сможете добавить настраиваемые события с реализацией, статические конструкторы и перегрузки операторов.

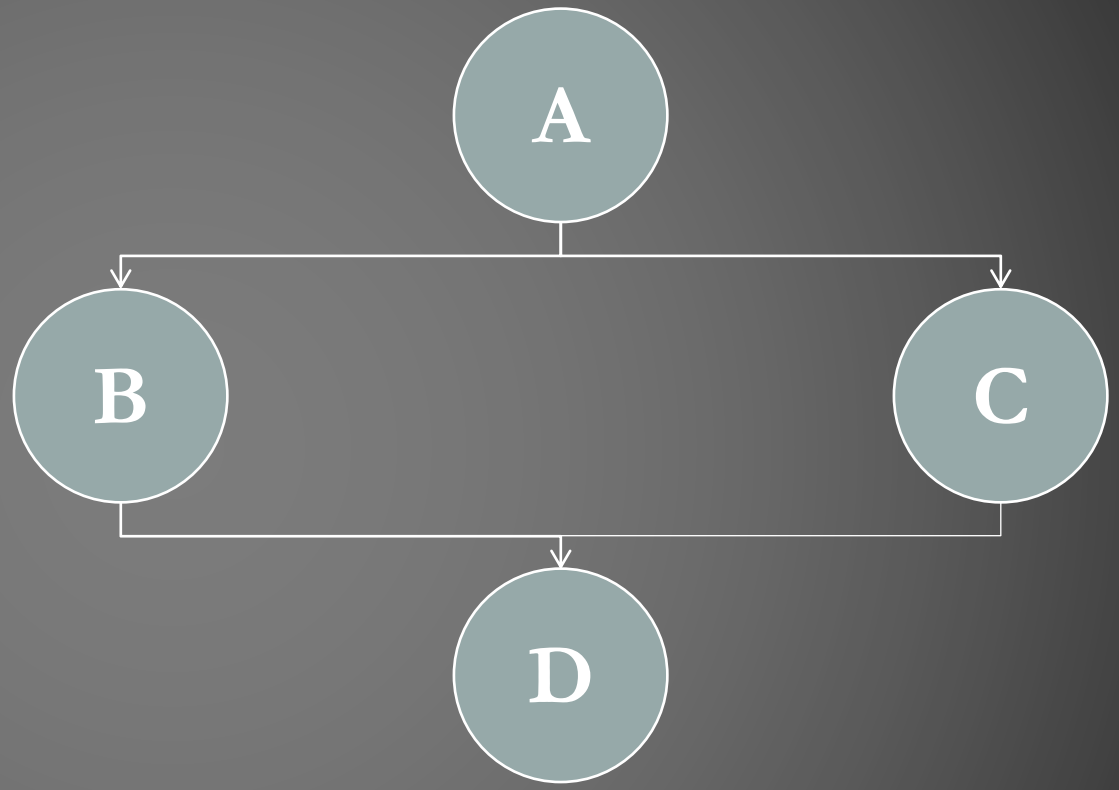
В первую очередь реализации методов по умолчанию добавляют более гибкие возможности обновления интерфейсов в библиотеках классов, позволяя определить дополнительные методы без вреда работоспособному коду.

При использовании модификатора доступа private интерфейс обязан предоставить реализацию соответствующего члена по умолчанию, любой не публичный член интерфейса должен реализовываться явно.

Имейте ввиду, что реализация интерфейса по умолчанию всегда является явной, по этой причине без переопределения вы не сможете вызвать соответствующие методы не по интерфейсной ссылке. Вы также можете добавлять реализацию по умолчанию свойствам или индексаторам, однако данные возможности используются редко.

Наследование Интерфейсов

Интерфейсы способны наследовать один или несколько других интерфейсов, в связи с чем может возникать вопрос: «А актуальна ли проблема ромбовидного наследования?»



И ответ на данный вопрос – нет, так как при реализации интерфейсов, имеющих общего родителя, вам достаточно определить одну реализацию метода, которая станет общей для всех интерфейсов. Такую реализацию принято называть **неявной**. Запомните, что по умолчанию мы реализуем интерфейсы именно неявно.

```
ссылка: 1
public interface ICreeper
{
    ссылка: 1
    void CreeperBoom(Object target);
}

ссылка: 1
public interface IChargedCreeper : ICreeper
{
    ссылка: 1
    void ChargedCreeperBoom(Object target, int strength);
}
Ссылок: 0
public class Creeper : IChargedCreeper
{
    // Реализация метода данного интерфейса
    ссылка: 1
    public void ChargedCreeperBoom(object target, int strength)
    {
        Console.WriteLine("The creeper goes boom-boom!");
        Console.WriteLine($"{target} receives {strength} damage.");
    }

    // Реализация метода родительского интерфейса обязательна!
    ссылка: 1
    public void CreeperBoom(object target)
    {
        Console.WriteLine($"You are going boom-boom, {target}!");
    }
}
```

Неявная Реализация Интерфейсов

При реализации интерфейса методы должны быть открытыми и не статическими, а сигнатура реализации в классе и объявления в интерфейсе должны совпадать.

Также важно понимать, что наследники классов, реализующих интерфейсы уже будут получать реализацию от своих родителей. В дополнение, при реализации интерфейса, наследующегося от какого-то другого интерфейса, Вы по сути реализуете оба интерфейса.

Вы можете пометить реализацию интерфейса в классе модификатором **virtual**, что позволит переопределять её в наследниках.

Ссылки Типов Интерфейсов

Как мы уже знаем, интерфейсы – ссылочные типы, вы всегда можете поместить любой объект типа, реализующего интерфейс по интерфейсной ссылке. При этом не забывайте, что для типов значений это будет означать упаковку. Тем не менее, это позволит вам крайне гибко оперировать с объектами и помещать по интерфейсным ссылкам объекты, никак не связанные друг с другом иерархией наследования, но при этом имеющие одинаковое поведение.

В дополнение, вы можете неявно приводить объекты к типам интерфейсов и явно в обратную сторону как при помощи оператора приведения типов **(T)**, так и через **as**.

Пример:

```
public static void Main() {
    IComparable<int> Iint = 4; // Упаковка производится, тип ссылочный
    IComparable Idouble = 2.5; // Упаковка тоже производится
    IEnumerable<int> IList = new List<int>();
    // Преобразование к типу интерфейса неявное
    ICollection<int> ILinked = new LinkedList<int>();
}
```

Совпадающие Члены Интерфейсов. Явная Реализация

C# допускает наличие методов с одинаковой сигнатурой в двух различных интерфейсах при их одновременной реализации классом или интерфейса, наследующегося от другого интерфейса с таким же по сигнатуре методом. При таком сценарии по умолчанию вы получите одинаковое поведение как в данном классе, так и по интерфейсным ссылкам.

Если вы хотите предоставить различную реализацию одного метода по ссылкам класса и интерфейсов, напишите **явную реализацию** методам. Для создания явной реализации необходимо без модификатора доступа после указания типа возвращаемого значения указать имя метода в формате **<Название Интерфейса>.<Имя Метода>(<Параметры>)**.

Важно понимать, что явно реализованные методы доступны только по ссылкам соответствующего типа интерфейса. Если вы реализуете метод интерфейса только явно, вы не сможете вызвать его из класса напрямую. Для устранения данной ситуации вы можете одновременно предоставить как явную, так и неявную реализацию интерфейса.

Пример Явной Реализации Интерфейсов

```
Ссылок: 5
public interface IKnight {
    Ссылок: 3
    bool Attack(Object Target);
}
Ссылок: 6
public interface IMage {
    Ссылок: 3
    void Attack(Object Target);
}
Ссылок: 2
public class RPGCharacter : IKnight, IMage {
    ссылка: 1
    public void Attack(object Target) {
        Console.WriteLine("Combining my skills...");
        (this as IKnight).Attack(Target);
        (this as IMage).Attack(Target);
    }
    Ссылок: 3
    bool IKnight.Attack(object Target) {
        Console.WriteLine($"Attacking {Target} with my spear.");
        if (Target is IMage)
            return true;
        return false;
    }
    Ссылок: 3
    void IMage.Attack(object Target) {
        Console.WriteLine($"Casting a spell on {Target}.");
    }
}
```

```
Ссылок: 0
class Program {
    Ссылок: 0
    static void Main() {
        RPGCharacter Refia = new RPGCharacter();
        IKnight knightForm = Refia as IKnight;
        IMage mageForm = Refia as IMage;
        Refia.Attack("Green Dragon");
        knightForm.Attack("Cyclops");
        mageForm.Attack("Darklegs");
    }
}
```

```
Ссылок: 3
public interface IInventory {
    // Индексатор с обоими аксессорами
    ссылка: 1
    Product this[int index] {
        get;
        set;
    }
    // Свойство только для чтения
    ссылка: 1
    Product Inventory {
        get;
    }
}
Ссылок: 10
public class Product {
    ссылка: 1
    public string ProductName { get; private set; }
    ссылка: 1
    public string Description { get; private set; }
    ссылка: 1
    public decimal BuyPrice { get; private set; }
    ссылка: 1
    public decimal SellPrice { get; private set; }
    ссылка: 1
    public Product(string name, string desc, decimal price) {
        ProductName = name;
        Description = desc;
        BuyPrice = price;
        SellPrice = price * 0.75m;
    }
}
Ссылок: 2
public class NPC {
    ссылка: 1
    public string Name { get; protected set; }
    ссылка: 1
    public string Type { get; protected set; }
    ссылка: 1
    public NPC(string name, string type) { Name = name; Type = type; }
}
```

Индексаторы и Свойства в Интерфейсах

Вы можете объявлять индексаторы и свойства в интерфейсах, однако существуют некоторые особенности:

- До версии C# 8.0 Вы не могли указать модификатор доступа ни индексатору, ни его аксессорам, они всегда были публичны.
- При неявной реализации свойств и индексаторов вы можете указать оба аксессора, даже если в интерфейсе объявлялся только один. Явная реализация должна совпадать с объявлением в интерфейсе.
- Объявление методов доступа get/set без тела не приводит к созданию автоматически реализованного свойства.

```
// Класс-наследник NPC, реализующий интерфейс IInventory
ссылка: 1
public class ShopNPC : NPC, IInventory {
    private Product[] sellingItems;
    // Явная реализация. Заметьте, что Вы можете обращаться к полям
    // по ссылке типа интерфейса.
    ссылка: 1
    Product IInventory.this[int index] {
        get => sellingItems[0];
        set => sellingItems[0] = value;
    }
    // Явная реализация свойства
    ссылка: 1
    Product IInventory.Inventory { get => new Product("Steam Sale", "50% off", 15.0m); }
    // Неявная реализация индексатора
    Ссылок: 0
    public Product this[int index] {
        get {
            if (index < 0 || index >= sellingItems.Length)
                throw new IndexOutOfRangeException("Invalid inventory index.");

            return sellingItems[index];
        }
        set {
            if (index < 0 || index >= sellingItems.Length)
                throw new IndexOutOfRangeException("Invalid inventory index.");

            sellingItems[index] = value;
        }
    }
    // Неявная реализация свойства. Обратите внимание, что неявная реализация может
    // определять дополнительный аксессор, которого не было в интерфейсе.
    Ссылок: 0
    public Product Inventory {
        get => sellingItems[0];
        set => sellingItems[0] = value;
    }
    Ссылок: 0
    public ShopNPC(string name, string type, uint products = 10) : base(name, type)
```

ПРИМЕР

```
public class Programmer : IComparable<Programmer>
{
    ссылка: 1
    public string Name { get; private set; }
    Ссылок: 3
    public int ExpInYears { get; private set; }
    Ссылок: 3
    public int ProjReleased { get; private set; }
    Ссылок: 5
    public string ProgLang { get; private set; }

    Ссылок: 0
    public int CompareTo(Programmer other)
    {
        if (ProgLang == "C#" && other.ProgLang != "C#") return 1;
        else if (ProgLang != "C#" && other.ProgLang == "C#") return -1;
        else return (ExpInYears * 10 + ProjReleased)
                .CompareTo(other.ExpInYears * 10 + ProjReleased);
    }

    Ссылок: 0
    public Programmer(string name, string programmingLang)
    {
        Name = name;
        ProgLang = programmingLang;
        ExpInYears = 0;
        ProjReleased = 0;
    }
}
```

IComparable<in T>

IComparable<T> – важный интерфейс пространства имён System, содержащий определение только 1 метода – **public int CompareTo(T other)**.

По соглашению **CompareTo(T other)** возвращает:

- Число больше нуля, если данный объект условно больше объекта other.
- Ноль, если объекты условно равны
- Число меньше нуля, если данный объект меньше объекта other.

Как правило, обычно данный интерфейс реализуется объектами, которые можно упорядочить или отсортировать. Например, метод Sort( ) класса Array выдаёт исключение как раз в случае, если хотя бы один из сортируемых объектов не реализует IComparable.

IComparer<in T> и Comparer<T>

IComparer<in T> – интерфейс, используемый для сравнения объектов типа T друг с другом в каком-либо вспомогательном классе. Предоставляет метод public int Compare(T x, T y), который по принципу работы схож с методом CompareTo( ) интерфейса IComparable<T>. Основное предназначение – создание нескольких сортировщиков для одного типа.

Обычно классы не реализуют IComparer<in T> напрямую, вместо этого рекомендуется наследоваться от абстрактного класса Comparer<T>, реализующего необобщённую версию интерфейса. Кроме реализации интерфейсов данный класс предлагает:

- Явную реализацию IComparer.Compare(object x, object y), сравнивает x с y и выбрасывает ArgumentException, если один из объектов не может быть преобразован к T или объекты не реализуют IComparable или IComparable<in T>.
- Свойство только для чтения **Comparer<T> Default**, возвращает компаратор по умолчанию (для IComparable<in T>) или результат вызова явной реализации метода IComparer.Compare.
- Абстрактный метод int Compare(T x, T y).
- Метод **public static Comparer<T> Create(Comparison<T> comparer)**, возвращающий новый компаратор на основе переданного делегата.

```
public class Programmer : ICloneable
{
    Ссылок: 3
    public string Name { get; private set; }
    Ссылок: 3
    public int ExpInYears { get; private set; }
    Ссылок: 3
    public int ProjReleased { get; private set; }
    Ссылок: 3
    public string ProgLang { get; private set; }

    Ссылок: 0
    public object Clone() {
        return new Programmer(Name, ProgLang, ExpInYears, ProjReleased);
    }

    Ссылок: 0
    public Programmer(string name, string progLang) {
        Name = name;
        ProgLang = progLang;
        ExpInYears = 0;
        ProjReleased = 0;
    }

    ссылка: 1
    private Programmer(string name, string progLang, int exp, int projects) {
        Name = name;
        ProgLang = progLang;
        ExpInYears = exp;
        ProjReleased = projects;
    }
}
```

ICloneable

ICloneable – интерфейс пространства имён System, содержащий определение только 1 метода – **public object Clone( )**.

Вы можете создавать как глубокие, так и поверхностные копии объектов при реализации метода Clone( ).

Напомним, что при глубоком копировании дублируются все объекты, при поверхностном – только копии верхнего уровня.

IEquatable<T>. Реализация Проверки на Равенство

Для того, чтобы полностью корректно иметь возможность проверять равенство/неравенство объектов, вы должны:

- Реализовать **IEquatable<T>** (метод **bool Equals<T>(T other)**) для проверки равенства объектов, т. к. именно данный метод используется коллекциями для проверок наподобие Contains, Remove, IndexOf в универсальных коллекциях.

- Перегрузить методы **Equals(object other)** и **GetHashCode( )** класса Object, при переопределении экземплярной версии также корректно будет работать и статический метод **Equals(object objA, object objB)**.

- Перегрузить операторы **==** и **!=**.

Задачи

Задание 1

Интерфейс может находится в (*укажите все верные ответы*):

1) Пространстве имён.
2) Статическом классе.
3) Глобальном пространстве имён.
4) Запечатанном классе.
5) Классе.

Задание 2

В результате выполнения фрагмента программы:

```
using System;

interface IInterface {
    void Meth(int a);
}

interface IInterface2 {
    void Meth(int a);
}

class Program : IInterface, IInterface2 {
    static void Main() {
        Meth(5);
    }

    public static void Meth(int a) {
        Console.Write(a * a - a / a % a);
    }
}
```

на экран будет выведено:

Примечание:
*Если возникнет ошибка компиляции, введите: ****
Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

Задание 3

В результате выполнения фрагмента программы:

```
using System;

interface IInterface {
    void Meth(int a);
}

interface IInterface2 {
    void Meth(int a);
}

class Program : IInterface, IInterface2 {
    static void Main() {
        ((IInterface2)(IInterface)new Program()).Meth(5);
    }

    void IInterface.Meth(int a) {
        Console.Write(a + a / a - a ^ a);
    }

    void IInterface2.Meth(int a) {
        Console.Write(a * a - a / a % a);
    }
}
```

на экран будет выведено:

Примечание:
*Если возникнет ошибка компиляции, введите: ****
Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

Задание 4

В результате выполнения фрагмента программы:

```
using System;

interface II {
    int Meth(int a, int b);
    int Prop { get; set; }
    int this[int x, int y] { get; set; }
}

class Program : II {
    int prop;

    public int this[int x, int y] {
        get => Prop % x * y;
        set => Prop = value * x % y;
    }

    public int Prop {
        get => prop - Meth(3, 4);
        set => prop = value * 3 + Meth(this[1, 1], this[2, 2]);
    }

    static void Main() {
        Program p = new Program();
        p[10, 15] = 7;
        p.Prop += 20;
        Console.Write(p[9, 3]);
    }

    public int Meth(int a, int b) => a + b + prop;
}
```

на экран будет выведено:

Примечание:

*Если возникнет ошибка компиляции, введите: ****

Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задачи

Задание 5

В интерфейсе может быть определено (*укажите все верные ответы*):

1) Событие.
2) Поле.
3) Свойство.
4) Индексатор.
5) Метод.

Ответы

Ответ 1: **12345**

Ответ 2: ***

Ответ 3: **24**

Ответ 4: **-21**

Ответ 5: **12345**