Основы программирования

Иерархия классов

Интерфейсы

Определение интерфейса

Интерфейс – это полностью абстрактный класс, содержащий набор семантически связанных абстрактных элементов (методов и свойств), предназначенных для реализации в наследующих классах.

Каждый класс, наследующий от интерфейса (говорят, что класс *реализует* интерфейс) должен самостоятельно реализовать каждый их унаследованных элементов.

Интерфейс …

- интерфейс -- ссылочный тип, который определяет набор методов и свойств, но не реализует их
- говорят, что наследники «реализуют» интерфейс
- один класс может реализовывать несколько интерфейсов
- объекты-наследники (классы и структуры) должны самостоятельно реализовать каждый из унаследованных элементов

...

- принято, что класс переопределяет ВСЕ элементы реализуемого интерфейса, хотя это и не обязательно
- если класс реализует не все унаследованные от интерфейса элементы, а такой класс обязан быть абстрактным, но это не делать так не рекомендуется

Элементы интерфейса

- методы, в том числе:
  - свойства;
  - индексаторы;
- события
- статические поля и константы (версия C# 8.0 и выше)

Интерфейсы не могут определять нестатические переменные.

Синтаксис интерфейса

```
<спецификатор доступа> interface <имя>
{
     <набор элементов>
}
```

Имя интерфейса: `I<сущностной корень>able`

Например: **`I`**`Compar`**`able`**, **`I`**`Clone`**`able`**.

...

- Рекомендуется описывать небольшое количество элементов, объединенных единой семантикой.

  Например, если класс реализует интерфейс `IComparable`, то его экземпляры умеют сравниваться между собой.

- Все элементы по умолчанию имеют спецификаторы `public abstract`.

Пример интерфейса IMovable

```
interface IMovable  {

// константа
const int minSpeed = 0;      // мин. скорость

// статическое поле
static int maxSpeed = 60;    // макс. скорость

// метод
void Move();                 // движение

// свойство
string Name { get; set; }    // название
}
```

Пример. Пояснения

- Определен интерфейс `IMovable`, представляющий движущийся объект, то есть описывает функционал, который должен быть у движущегося объекта
- Методы и свойства интерфейса не имеют реализации (сравните с абстрактными методами и свойствами).
- Интерфейс определяет метод `Move`, представляющий некоторое передвижение. Свойство `Name` не является автоматическим свойством.

Пример. Пояснения

- Если члены интерфейса - методы и свойства – не имеют модификаторов доступа, но фактически по умолчанию доступ `public`, т.к. цель интерфейса – определение функционала для реализации его классом.
- Констант и статические переменные, которые в классах и структурах по умолчанию имеют модификатор `private`, в интерфейсах же они имеют по умолчанию модификатор `public`.

Особенности интерфейсов в C# 8.0

Начиная с версии C# 8.0 интерфейсы поддерживают реализацию методов и свойств по умолчанию.

Это значит, что мы можем определить в интерфейсах полноценные методы и свойства, которые имеют реализацию как в обычных классах и структурах.

Реализация метода интерфейса в C# 8.0

```
interface IMovable    {

// реализация метода по умолчанию

void Move()           {
  Console.WriteLine("Двигаюсь");
}

}
```

Реализация по умолчанию. Для чего?

Допустим, у нас есть куча классов, которые реализуют некоторый интерфейс.

Если мы добавим в этот интерфейс новый метод, то мы будем обязаны реализовать этот метод во всех классах, реализующих данный интерфейс.

Теперь вместо реализации метода во всех классах нам достаточно определить его реализацию по умолчанию в интерфейсе.

Если класс не реализует метод, будет применяться реализация по умолчанию.

Пример

```
class Person : IMovable { }

class Car : IMovable        {

public void Move()          {
Console.WriteLine("Driving");
}

}
```

Пример

```
class Program    {
static void Main(string[] args)       {
IMovable tom = new Person();
Car tesla = new Car();
tom.Move();      // Walking
tesla.Move();    // Driving
}
}
```

А так нельзя:

```
Person tom = new Person();
tom.Move(); // Ошибка - метод Move не определен
```

Реализация свойства C# 8.0

В интерфейсах нельзя определять нестатические переменные, то есть в свойствах интерфейса нельзя манипулировать состоянием объекта.

```
interface Imovable    {
void Move()      {
  Console.WriteLine("Двигаюсь");
}

//реализация свойства по умолчанию
int MaxSpeed { get { return 0; } }

}
```

Модификатор private в интерфейсах в C# 8.0

Если интерфейс имеет закрытые методы и свойства (с модификатором `private`), то они должны иметь реализацию по умолчанию.

То же самое относится к любым статическим методам и свойствам (не обязательно закрытым).

Модификатор private в интерфейсах в C# 8.0

```
interface IMovable
{
public const int minSpeed = 0;       // мин. скорость
private static int maxSpeed = 60;    // макс.скорость

// время, за которое надо пройти расстояние distance
со скоростью speed
static double GetTime(double distance, double speed)
(return distance / speed;)

static int MaxSpeed {
get { return maxSpeed; }
set { if (value > 0) maxSpeed = value;  }
}
}
```

Модификатор private в интерфейсах в C# 8.0

```
class Program     {

static void Main(string[] args)          {
  Console.WriteLine(IMovable.MaxSpeed);
  IMovable.MaxSpeed = 65;
  Console.WriteLine(IMovable.MaxSpeed);
  double time = IMovable.GetTime(100, 10);
  Console.WriteLine(time);
}

}
```

Модификаторы доступа интерфейсов

- Как и классы, интерфейсы по умолчанию имеют уровень доступа `internal` – такой интерфейс доступен только в рамках текущего проекта.
- С помощью модификатора `public` можно сделать интерфейс общедоступным:

```
public interface IMovable      {
void Move();
}
```

Пример. Интерфейс

```
public interface ISumable
{
      int Sum();
}
```

Пример. Class1

```
public class Class1 : ISumable      {
int a;     int b;

public Class1(int a, int b)          {
this.a = a;  this.b = b;
}

public int Sum()     {
return a + b;
}
}
```

Пример. Class2

```
public class Class2 : ISumable      {
int a;     int b;     int c;

public Class2(int a, int b, int c) {
this.a = a;  this.b = b;  this.c = c;
}

public int Sum()     {
return a + b + c;
}
}
```

Пример. Проверяем работу

```
…
Class1 class1 = new Class1(1,2);
Class2 class2 = new Class2(3,4,5);

ISumable i = class1;
Console.WriteLine("Сумма: " + i.Sum());
i = class2;
Console.WriteLine("Сумма: " + i.Sum());
…
```

```
Сумма: 3
Сумма: 12
```

Пример. Пояснения

- результат работы метода зависит от реального типа объекта, вызывающего метод на исполнение, а не от типа ссылки на объект.
- под однотипными интерфейсными ссылками могут храниться объекты различных реальных классов.
- метод, объявленный в интерфейсе, вызывается одинаково (одинаково внешнее представление), а исполняется по-разному (различна внутренняя реализация в разных классах).

Пример. Продолжение

```
…
ISumable[] mas = new ISumable[3];
mas[0] = new Class1(10, 20);
mas[1] = new Class2(1, 2, 3);
mas[2] = new Class2(4, 5, 6);

foreach (ISumable k in mas)
{
Console.WriteLine("Сумма: " + k.Sum());
}
…
```

```
Сумма: 30
Сумма: 6
Сумма: 15
```

Пример. Пояснения

Несмотря на то, что объекты `mas[0]` и `mas[1]` на самом деле принадлежат разным классам,

если рассматривать их с точки зрения их интерфейса, то они одинаковы,

значит их можно хранить в массиве и обращаться к ним единым образом.

Интерфейсные ссылки

Получение интерфейсной ссылки на реальный объект возможно несколькими способами:

1)присвоение ссылке интерфейсного типа ссылки на реальный объект;

2)использование операции приведения типа `as`;

3)использование операции проверки принадлежности типу `is`.

Интерфейсные ссылки. Способ 1

```
Class1 class1 = new Class1(1,2);
ISumable i = class 1;
```

Аналогично

```
Class1 class1 = new Class1(1,2);
ISumable i = (ISumable)class1;
```

Если явное и неявное приведение типа невозможно, то происходит выброс исключения `InvalidCastException`.

Интерфейсные ссылки. Способ 2

Использование операции проверки принадлежности типу `is`.

Операция возвращает `true`, если объект можно преобразовать к заданному типу, и `false`, если нельзя.

```
if (class1 is ISumable)
ISumable i = class1;
//или так ISumable i = (ISumable)class1;
else …
```

Ссылки типа интерфейс могут использоваться в качестве параметров в методы и возвращаться как результат.

Интерфейсные ссылки. Способ 3

Операция `as` выполняет преобразование к заданному типу, а если это невозможно, то возвращает `null`.

```
ISumable i = class1 as ISumable;
```

Результат нужно проверить:

```
if (i !== null) i.Sum()
else …
```

Множественная реализация интерфейсов

В языке C# поддерживается множественное наследование от интерфейсов -- один класс может реализовывать несколько интерфейсов.

Может возникнуть ситуация, когда в реализуемых классом интерфейсах есть методы, точно совпадающие по сигнатуре.

Явная реализация интерфейса

При явной реализации указывается название метода или свойства вместе с названием интерфейса, при этом мы не можем использовать модификатор `public`, то есть методы являются закрытыми:

```
interface IActionable {
void Move();
}
class BaseAction : IActionable   {
void IActionable.Move()          {
Console.WriteLine("Двигаюсь (класс BaseAction)");
}
}
```

Явная реализация интерфейса

При явной реализации интерфейса его методы и свойства не являются частью интерфейса класса. Поэтому напрямую через объект класса мы к ним обратиться не сможем:

См. пример на следующем слайде.

Явная реализация интерфейса

```
static void Main(string[] args)  {

BaseAction action = new BaseAction();
(IActionable)action.Move();
            // необходимо приведение к типу IAction


// или так
IAction action2 = new BaseAction();
action2.Move();

Console.ReadKey();
}
```

Явная реализация интерфейса

Это может понадобиться когда класс реализует несколько интерфейсов, но они имеют один и тот же метод с одним и тем же возвращаемым результатом и одним и тем же набором параметров.

Тогда явная реализация позволяет различить реализуемые интерфейсы.

Подробнее:
`https://metanit.com/sharp/tutorial/3.44.php`

Интерфейсы и наследование

Один и тот же интерфейс могут реализовывать классы, связанные наследованием. То есть класс-предок и класс-потомок реализует один и тот же интерфейс.

Возможны варианты:

1. Не переопределять методы интерфейса в классе-потомке.
   Тогда их реализация наследуется из класса-предка.
2. Реализовать в классе-потомке методы интерфейса.
   Тогда их реализации, унаследованные от класса-предка, игнорируются. Этот вариант предпочтительней.

...

Подробнее:
C# и .NET | Реализация интерфейсов в базовых и производных классах (metanit.com)
https://metanit.com/sharp/tutorial/3.47.php

Если сигнатуры методов совпадают …

Может быть принято решение о единственной реализации классом обоих методов.

В классе описывается одна реализация метода.

Именно эта реализация и будет вызываться во всех случаях, независимо от типа ссылки на реальный объект:

– ссылки типа первый интерфейс,
– ссылки типа второй интерфейс или
– ссылки реального типа.

Если сигнатуры методов совпадают …

Бывают ситуации, когда семантика методов может отличаться: реализация должна быть различной.

В этом случае применяется явная реализация интерфейсов: в классе в заголовке метода явно указывается, какому интерфейсу будет принадлежать реализация метода.

Пример. Интерфейсы

```
public interface ISumable
{
int Sum();
}

public interface IDblSumable
{
int Sum();
}
```

Пример. Класс

```
public class MyClass: ISumable, IDblSumable {
int a; int b;

public MyClass(int a, int b)      {
this.a = a; this.b = b;                  }

public int ISumable.Sum()         {
return a + b;                            }

public int IDblSumable.Sum()      {
return a*a + b*b;                        }
      }
```

Пример. Проверка

```
…
MyClass i = new MyClass(1,2);
ISumable isum = i;
Console.WriteLine("Сумма: "+isum.Sum());
IDblSumable isqr = i;
Console.WriteLine("Сумма: "+isqr.Sum());
//Console.WriteLine("Сумма: "+i.Sum());
…
```

```
Сумма: 3
Сумма: 5
```

Пример. Пояснения

При явной реализации интерфейса соответствующие версии вызываются только тогда, когда на объект указывает ссылка соответствующего интерфейсного типа

```
//Console.WriteLine("Сумма: "+i.Sum());
```

-- ошибка компиляции, так как метода `Sum`, принадлежащего самому классу `MyClass`, не существует.

Наследование интерфейсов

1. Интерфейс – это класс и он может быть включен в иерархию наследования и как предок, и как потомок.
2. Интерфейс не может наследовать от класса.
3. Для интерфейсов в языке определено множественное наследование.
4. В интерфейсе-потомке содержатся все методы, унаследованные от интерфейсов-предков.

Наследование интерфейсов

5. Новый метод взамен унаследованного нужно описывать с помощью ключевого слова `new`.
6. Чтобы обратиться к скрытому теперь методу интерфейса-предка необходимо явное преобразование к типу интерфейса-предка.
7. Наследующий интерфейс не может расширять область видимости базового интерфейса.

Наследование интерфейсов нужно?

На данном этапе изучения особого смысла в наследовании интерфейсов от интерфейсов ВООБЩЕ-ТО НЕТ.

Так как при наследовании интерфейсов наследуются только заголовки методов, а не их реализации, то есть наследуется внешний интерфейс

Добавление интерфейса в проект

Для добавления интерфейса в проект можно нажать правой кнопкой мыши на проект в «Обозревателе решений» и в появившемся контекстном меню выбрать

`Add-> New Item`... и в диалоговом окне добавления нового компонента выбрать пункт `Interface`.