# Разработка .NET приложений

Лекция 4

Интерфейсы

# Сегодня

- Интерфейсы. Классическое представление, до C# 8
- Реализация методов интерфейсов по умолчанию (C# 8.0 и позднее)

# Интерфейсы

- Интерфейс - контракт, поведение, которое реализующий класс обязуется поддерживать.
- Набор семантически связанных абстрактных членов.
- Интерфейс определяет спецификацию, но не определяет реализацию *
- Класс – наследуют, а интерфейс – реализуют

* - до C#8, .NET Core 3

# Объявление интерфейса

Объявление интерфейса

```
[attributes] [modifiers] interface interface_name
    [: base_interfaces]
{
  interface_body
}
```

Пример объявления интерфейса

```
public interface IDisposable {
   void Dispose();
}

public interface IPointy {
   byte Points {get;}
   int GetNumberOfPoints();
}
```

# Члены интерфейса

- Интерфейс может содержать следующие члены
  - Методы
  - Свойства
  - Индексаторы
  - События
- Не могут содержать поля *
- Не могут содержать реализацию *
- Члены интерфейса не могут быть статическими *
- Члены интерфейса не могут иметь модификаторов *

  Члены интерфейса всегда public

* до C#8, .NET Core 3

# Реализация интерфейса

- Классы и структуры могут реализовать интерфейсы
  - public class Circle2d : Ellipse2d, ICloneable { ….}
- Базовый класс должен идти первым в списке, а за ним список реализуемых интерфейсов.
- Можно реализовать одновременно несколько интерфейсов
- Класс, реализующий интерфейс *:
  - Либо реализует все его члены
  - Либо объявляет их (и себя) абстрактными

* до C#8, .NET Core 3

# Явная и неявная реализация

- **Неявная реализация интерфейса**
- Записывается короткое (как обычно) имя каждого метода, свойства и т.д.

```
public object Clone() { … }
```

- Доступность члена извне:
  - Метод доступен как обычны член типа
    `peremen.Clone()`
  - Метод доступен и через интерфейс
  - `((ICloneable)peremen).Clone()`
  - `IDisposable classForDispose = (IDisposable)peremen;`
  - `classForDispose.Dispose()`

- **Явная (explicit) реализация интерфейса**
- Записывается квалифицированное имя каждого метода, свойства и т.д. (т.е. с указанием интерфейса)
  - `object ICloneable.Clone() { … }`
- Доступность члена извне:
  - Метод доступен <u>только</u> через интерфейс
    `((ICloneable)peremen).Clone()`
- Используется при наличии конфликта
- Не указывается модификатор доступа. Он неявно **public**

# Привязка методов интерфейса

- Привязка к не виртуальным методам
  - Производному классу, чтобы сменить реализацию, требуется самому реализовать интерфейс
- Привязка к виртуальным методам
  - Производным классам достаточно просто переопределить виртуальный метод, чтобы сменить реализацию

# Об интерфейсах

- Интерфейсы могут наследоваться от других интерфейсов.
  - Производный интерфейс включает в себя все из базовых интерфейсов
  - Тип (класс, структура), реализующие производный интерфейс должен реализовать также и все базовые интерфейсы
  - public interface IList : ICollection, IEnumerable {….}
- Переменные интерфейсного типа
  - Могут принимать значения любого типа, реализующего интерфейс
  - IEnumerabe sequence = new int[] {1,2,3,4,5}
- Передача интерфейсов в качестве параметров
  - Метод принимает любой тип, реализующий данный интерфейс
  - public void GetShape(IDrawable  shape);
- Интерфейсы в качестве возвращаемых параметров
  - Метод возвращает некий тип, который реализует данный интерфейс
  - public IDrawable GetShape();
- В **.NET** все существующие интерфейсы начинаются с **I**

# Абстрактные классы и интерфейсы

- Абстрактные классы и интерфейсы обеспечивают абстракцию
- Абстрактные классы могут содержать реализацию, интерфейсы – нет *
- Интерфейсы могут содержать только методы, свойства, события и индексаторы
- Все члены интерфейса – public. Это открытый контракт
- Используйте абстрактные классы, если:
  - Требуется определить определенную функциональность, которую могут использовать все наследники
  - Производный класс “является” базовым
- Используйте интерфейсы
  - Производный класс “реализует” интерфейс
  - Требуется множественное наследование

* до C#8, .NET Core 3

# Примеры интерфейсов

- ICloneable
  - Определяет единственный метод object Clone()
  - Позволяет выполнять копирование объектов
- IDisposable
  - Определяет единственный метод Dispose()
  - Реализуйте этот метод, если требуется освобождение внешних ресурсов
    - Файлы, соединения и т.д.
- IComparable – используется при сортировках при сравнении 2 элементов
- IEnumerable, IEnumerable<T>, IEnumerator, IEnumerator<T>
  - Позволяют итерировать по коллекции
- IDictionary<TKey, TValue>, IList<T>, ICollection<T>
  - Интерфейсы реализуемые коллекциями

# Демонстрации

Интерфейсы

# Сегодня

- Интерфейсы. Классическое представление, до C# 8
- Реализация методов интерфейсов по умолчанию (C# 8.0 и позднее)

# Реализация методов интерфейсов по умолчанию

- Проблема:
  - Выпущенный интерфейс тяжело изменить, расширить его
  - При расширении интерфейса нужно дореализовать функциональность, добавленную в интерфейс во всех потребителях

- В **C# 8** (в **.NET Core 3**, нет в **.NET Framework**) добавили возможность добавлять методы и свойства с реализацией

```
public interface ICalculator
  {
    int Add(int x, int y);
    int Sub(int x, int y);

    int Mul(int x, int y)
    {
      return x * y;
    }
  }
```

```
public class Calculator : ICalculator
  {
    public int Add(int x, int y)
    {
      return x + y;
    }

    public int Sub(int x, int y)
    {
      return x - y;
    }
}
```

```
Calculator calculator = new Calculator();

int res = calculator.Add(2,5);
 int res = calculator.Sub(2,5);

 int res = ((ICalculator)calculator).Mul(2,5);
```

Использовать можно только через интерфейс
Нет проблемы множественного наследования

# Переопределение реализации по умолчанию

```
public interface ICalculator
  {
    int Add(int x, int y);
    int Sub(int x, int y);

    int Mul(int x, int y)
    {
      return x * y;
    }
  }
```

```
public class Calculator : ICalculator
  {
    public int Add(int x, int y)
    {
      return x + y;
    }

    public int Sub(int x, int y)
    {
      return x - y;
    }

    public int Mul(int x, int y)
    {
      return x * y % 4;
    }
}
```

```
Calculator calculator = new Calculator();

int res = calculator.Add(2,5);
 int res = calculator.Sub(2,5);

 int res = ((ICalculator)calculator).Mul(2,5);

int res = calculator.Mul(2,5);
```

Использовать можно напрямую и через интерфейс

Реализация будет одинаковая

# Статические члены в интерфейсе

- Можно заводить статические члены
  - Поля
  - Константы
  - Методы
  - Свойства
- Статические члены обязаны иметь в интерфейсе реализацию по умолчанию
- Не могут содержать инстансных (не статических) полей

```
public interface ICalculator
  {
    static int Module = 10;
    int Add(int x, int y);

    int Mul(int x, int y)
    {
      return x * y % Module;
    }
  }
```

```
Calculator calculator = new Calculator();

ICalculator.Module = 12;

Console.WriteLine(((ICalculator)calculator).Mul(2,5));
```

Обращение к статическим членам только через интерфейс (как и с обычными типам, обращение к статическим членам только через указание типа)

# Модификаторы доступа в интерфейсе

- По умолчанию все члены в интерфейсе **public**
- Могут быть заведены **private** члены, доступ к которым возможен только из реализации членов внутри интерфейса

```
public interface ICalculator
  {
     private const double Pi = 3.1415;
     static double GetPi() => Pi; // статический метод с реализацией по умолчанию

    private static int _module = 10;
    public int MulModule { get => _module; set =>_module = value; } // свойство с реализацией по умолчанию

    public int DevModule { get; set; } // свойство без реализации, реализующее интерфейс классы обязаны реализовать это свойство

    int Add(int x, int y);
     int Sub(int x, int y);

     int Mul(int x, int y) // метод с реализацией по умолчанию
    {
       return x * y % _module;
     }
  }
```

# Члены интерфейса в C# 8 и позднее

- Интерфейс может содержать следующие члены
  - Методы
  - Свойства
  - Индексаторы
  - События
- ~~Не могут содержать поля~~ * Могут содержать статические поля и константы
- ~~Не могут содержать реализацию~~ * Могут содержать реализацию по умолчанию
- ~~Члены интерфейса не могут быть статическими~~ * Члены могут быть статическими
- ~~Члены интерфейса не могут иметь модификаторов~~ * Члены могут иметь модификаторы доступа

  Члены интерфейса ~~всегда~~ по умолчанию public

- Не добавляют проблем множественного наследования
- Позволяют расширять уже существующие интерфейсы

# Сегодня

- Интерфейсы. Классическое представление, до C# 8
- Реализация методов интерфейсов по умолчанию (C# 8.0 и позднее)