

Модуль 9:

Наследование в C#

Обзор

- Производные классы
- Реализация методов
- Изолированные классы
- Использование интерфейсов
- Абстрактные классы

◆ Производные классы

- **Расширение базовых классов**
- **Доступ к членам базового класса**
- **Вызов конструкторов базового класса**

Расширение базовых классов

■ Синтаксис наследования от базового класса

```
class Token
```

```
{
```

```
...
```

```
}
```

```
class CommentToken: Token
```

```
{
```

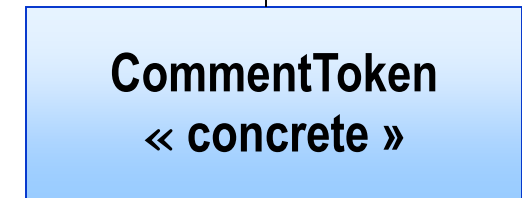
```
...
```

```
}
```

Производный класс

Базовый класс


Двоеточие




- Производный класс наследует большую часть элементов базового класса
- Производный класс не может быть более доступным, чем базовый

Доступ к членам базового класса

```
class Token
{
    ...
    protected string name;
}
class CommentToken: Token
{
    ...
    public string Name( )
    {
        return name;
    }
}
```



```
class Outside
{
    void Fails(Token t)
    {
        ...
        t.name
        ...
    }
}
```



- Унаследованные **protected** члены неявно становятся **protected** членами для будущего производного класса
- Методы производного класса имеют доступ только к собственным **protected** членам
- Нельзя использовать модификатор доступа **protected** для структур

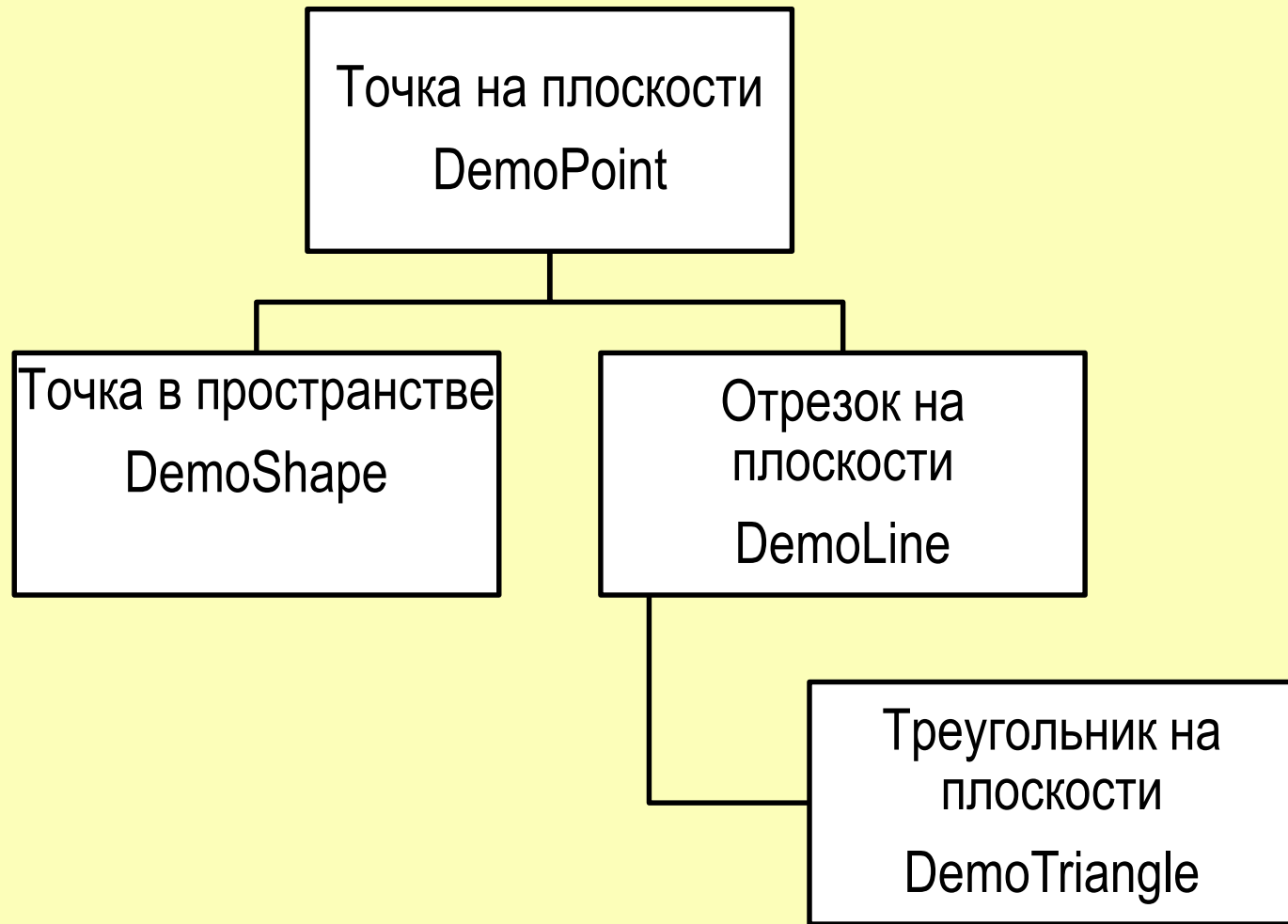
Вызов конструкторов базового класса

- При объявлении конструктора необходимо использовать ключевое слово **base**

```
class Token
{
    protected Token(string name) { ... }
    ...
}
class CommentToken: Token
{
    public CommentToken(string name) : base(name) { }
    ...
}
```

- Закрытый конструктор базового класса не доступен в производном классе
- Для доступа к скрытому имени используйте ключевое слово **base**

Многоуровневая иерархия. Пример



◆ Работа с методами в производных классах

- Определение виртуальных методов
- Применение виртуальных методов
- Переопределение методов
- Применение переопределенных методов
- Использование new для сокрытия методов
- Применение ключевого слова new
- Практика: Работа с методами в производных классов
- Тест: Найдите ошибки

Определение виртуальных методов

- Синтаксис: При объявлении используйте ключевое слово **virtual**

```
class Token
{
    ...
    public int LineNumber( )
    { ...
    }
    public virtual string Name( )
    { ...
    }
}
```

- Виртуальные методы реализуют полиморфизм

Применение виртуальных методов

- Правила определения виртуальных методов:
 - Вы не можете определить виртуальный метод как статический
 - Нельзя определять виртуальный метод как **private**

Переопределение методов

- Синтаксис: При объявлении используйте ключевое слово **override**

```
class Token
{
    ...
    public virtual string Name( ) { ... }
}
class CommentToken: Token
{
    ...
    public override string Name( ) { ... }
}
```

Применение переопределенных методов

- Прототипы виртуального и переопределенного методов должны полностью совпадать

```
class Token
{
    ...
    public int LineNumber( ) { ... }
    public virtual string Name( ) { ... }
}
class CommentToken: Token
{
    ...
    public override int LineNumber( ) { ... }
    public override string Name( ) { ... }
}
```



- Можно переопределять только соответствующие виртуальные методы
- Вы можете переопределять переопределенные методы
- Вы не можете явно определить метод со спецификатором **override** как **virtual**
- Нельзя определять переопределенные методы как статические или закрытые

Использование new для сокрытия методов

- Синтаксис: Для сокрытия метода используйте ключевое слово **new**

```
class Token
{
    ...
    public int LineNumber( ) { ... }
}
class CommentToken: Token
{
    ...
    new public int LineNumber( ) { ... }
}
```

Применение ключевого слова new

- Можно скрыть как виртуальный, так и обычный метод

```
class Token
{
    ...
    public int LineNumber( ) { ... }
    public virtual string Name( ) { ... }
}
class CommentToken: Token
{
    ...
    new public int LineNumber( ) { ... }
    public override string Name( ) { ... }
}
```

- Разрешает конфликты имен в коде
- Скрывает метод, имеющий такую же сигнатуру

Использование ключевого слова sealed

- От sealed-класса нельзя наследовать
- В .NET Framework есть много классов, для которых запрещено наследование: String, StringBuilder и т.д.
- Синтаксис: Используйте ключевое слово sealed

```
namespace System
{
    public sealed class String
    {
        ...
    }
}
namespace Mine
{
    class FancyString: String { ... } ✖
}
```

Тест: Найдите ошибки

```
class Base
{
    public void Alpha( ) { ... }
    public virtual void Beta( ) { ... }
    public virtual void Gamma(int i) { ... }
    public virtual void Delta( ) { ... }
    private virtual void Epsilon( ) { ... }
}
class Derived: Base
{
    public override void Alpha( ) { ... }
    protected override void Beta( ) { ... }
    public override void Gamma(double d) { ... }
    public override int Delta( ) { ... }
}
```


◆ Интерфейсы

- **Объявление интерфейсов**
- **Реализация нескольких интерфейсов**
- **Реализация методов интерфейса**
- **Реализация методов интерфейса явным образом**
- **Тест: Найдите ошибки**

Объявление интерфейсов

- Синтаксис: При объявлении используйте ключевое слово **interface**

Имена интерфейсов должны начинаться с заглавной "I"

```
interface IToken  
{  
    int LineNumber( );  
    string Name( );  
}
```

IToken « interface »
LineNumber() Name()

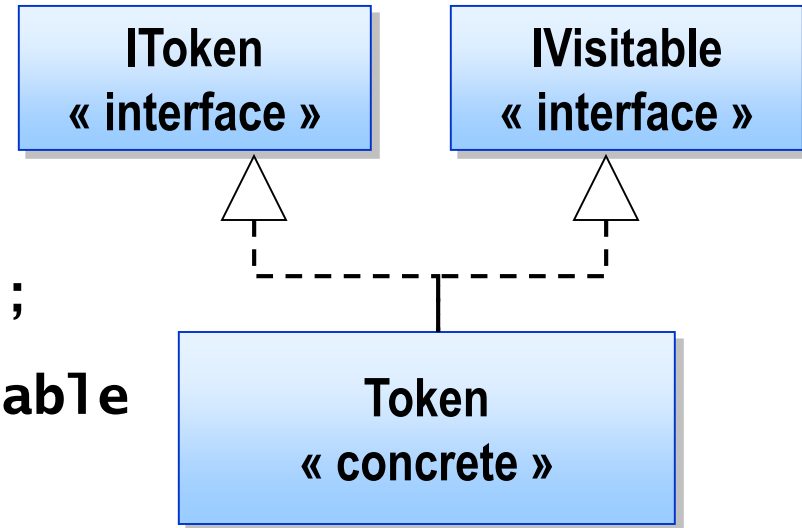
Нет модификаторов доступа

Нет тела метода

Реализация нескольких интерфейсов

- Класс может реализовывать несколько интерфейсов

```
interface IToken
{
    string Name( );
}
interface IVisitable
{
    void Accept(IVisitor v);
}
class Token: IToken, IVisitable
{
    ...
}
```



- Интерфейс может наследоваться от нескольких интерфейсов
- Класс может быть более доступным, чем его базовый интерфейс
- Интерфейс не может быть более доступным, чем его базовые интерфейсы
- Класс должен реализовывать все унаследованные от интерфейса методы

Реализация методов интерфейса

- Прототип реализации метода должен полностью совпадать с прототипом метода интерфейса
- Метод, реализующий интерфейс может быть как виртуальным так и невиртуальным

```
class Token: IToken, IVisitable
{
    public virtual string Name( )
    { ...
    }
    public void Accept(IVisitor v)
    { ...
    }
}
```

Тот же модификатор доступа
Тот же тип возвращаемых значений
То же имя
Те же параметры

Реализация методов интерфейса явным образом

- Указывайте полное имя метода, включающее имя интерфейса

```
class Token: IToken, IVisitable
{
    string IToken.Name( )
    { ...
    }
    void IVisitable.Accept(IVisitor v)
    { ...
    }
}
```

- Ограничения при реализации методов явным образом
 - Доступ к методу только через ссылку на интерфейс
 - Нельзя объявить метод как виртуальный
 - Нельзя указывать модификатор доступа

Тест: Найдите ошибки

```
interface IToken
{
    string Name( );
    int LineNumber( ) { return 42; }
    string name;
}

class Token
{
    string IToken.Name( ) { ... }
    static void Main( )
    {
        IToken t = new IToken( );
    }
}
```

◆ Абстрактные классы

- **Объявление абстрактных классов**
- **Использование абстрактных классов в классовой иерархии**
- **Сравнение абстрактных классов с интерфейсами**
- **Реализация абстрактных методов**
- **Применение абстрактных методов**
- **Тест: Найдите ошибки**

Объявление абстрактных классов

- Используйте ключевое слово **abstract**

```
abstract class Token
{
    ...
}
class Test
{
    static void Main( )
    {
        new Token( );
    }
}
```

Token
{ abstract }

Нельзя создавать экземпляры
абстрактного класса

Использование абстрактных классов в классовой иерархии

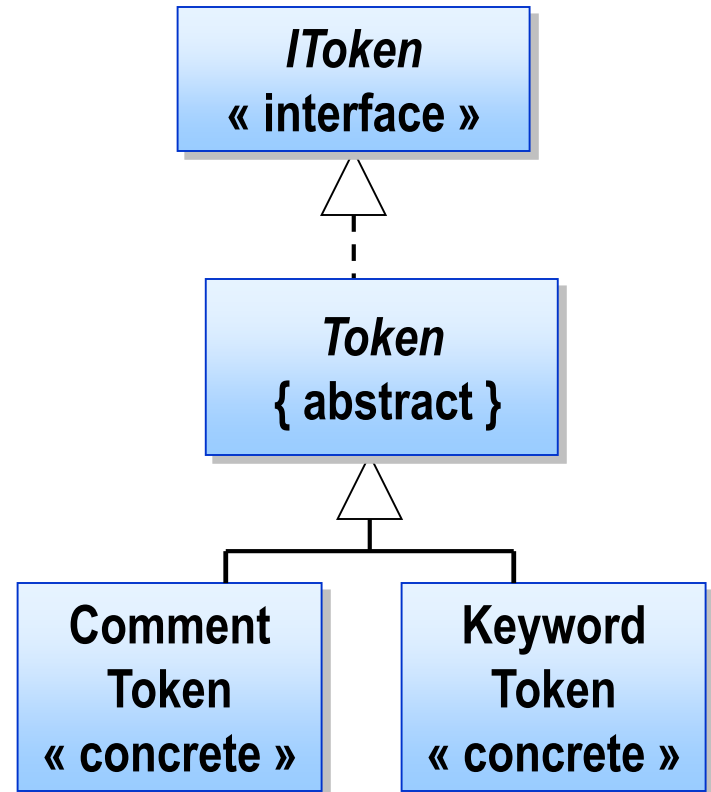
■ Пример 1

```
interface IToken
{
    string Name( );
}

abstract class Token: IToken
{
    string IToken.Name( )
    {
        ...
    }
    ...
}

class CommentToken: Token
{
    ...
}

class KeywordToken: Token
{
    ...
}
```



Использование абстрактных классов в классовой иерархии (продолжение)

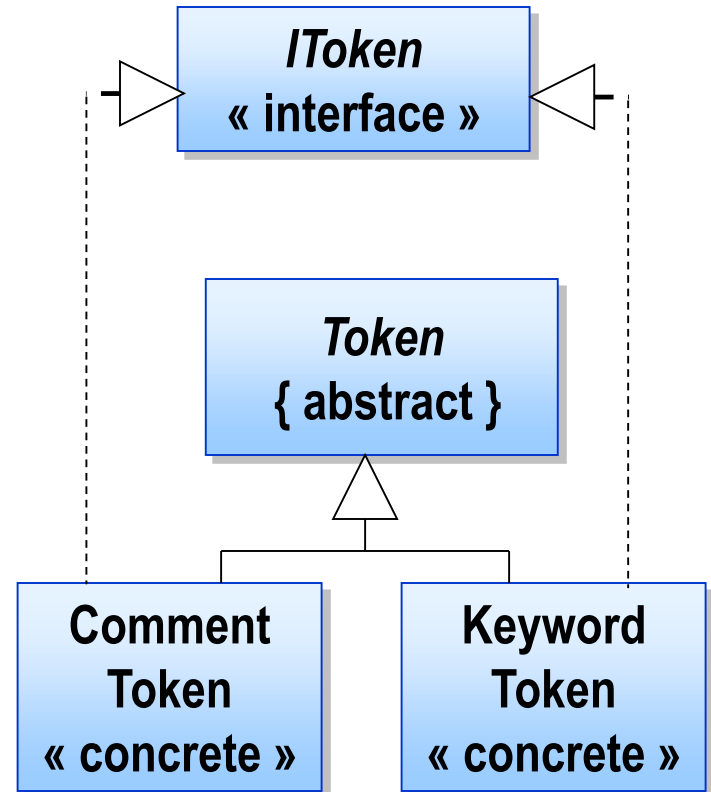
■ Пример 2

```
interface IToken
{
    string Name( );
}

abstract class Token
{
    public virtual string Name( )
    {
        ...
    }
    ...
}

class CommentToken: Token, IToken
{
    ...
}

class KeywordToken: Token, IToken
{
    ...
}
```



Сравнение абстрактных классов с интерфейсами

■ Сходства

- Нельзя создавать их экземпляры
- Нельзя объявлять с ключевым словом `sealed`

■ Различия

- Интерфейсы не содержат тела с реализацией
- Все члены интерфейса имеют модификатор `public`
- Интерфейсы могут наследоваться только от интерфейсов

Реализация абстрактных методов

- Синтаксис: Используйте ключевое слово **abstract**

```
abstract class Token
{
    public virtual string Name( ) { ... }
    public abstract int Length( );
}
class CommentToken: Token
{
    public override string Name( ) { ... }
    public override int Length( ) { ... }
}
```

- Абстрактные методы можно объявлять только в абстрактных классах
- Абстрактные методы не содержат реализации

Применение абстрактных методов

- Абстрактные методы виртуальны
- Можно переопределить абстрактные методы в производных классах
- Абстрактные методы могут переопределять виртуальные методы базового класса
- Абстрактные методы могут переопределять методы базового класса, объявленные как `override`

Тест: Найдите ошибки

```
class First
{
    public abstract void Method( );
}
```

1

```
abstract class Second
{
    public abstract void Method( ) { }
}
```

2

```
interface IThird
{
    void Method( );
}
abstract class Third: IThird
{
}
```

3

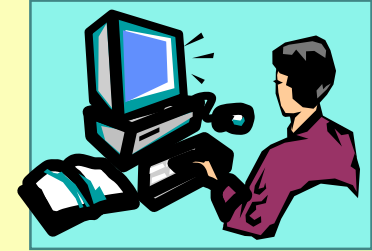
Практика: Работа с методами в производных классах

```
class A {  
    public virtual void M() { Console.Write("A"); }  
}  
class B: A {  
    public override void M() { Console.Write("B"); }  
}  
class C: B {  
    new public virtual void M() { Console.Write("C"); }  
}  
class D: C {  
    public override void M() { Console.Write("D"); }  
    static void Main() {  
        D d = new D(); C c = d; B b = c; A a = b;  
        d.M(); c.M(); b.M(); a.M();  
    }  
}
```

Лабораторная работа 9: Использование наследования при реализации интерфейсов

Дополнительно:

- Создать абстрактный класс *Figure* с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре.
- Создать производные классы: *Rectangle* (прямоугольник), *Circle* (круг), *Triangle* (треугольник) со своими методами вычисления площади и периметра.
- Создать массив *n* фигур и вывести полную информацию о фигурах на экран.



Методические указания:

- Полную структуру классов и их взаимосвязь продумать самостоятельно.
- Для абстрактного класса определить какие методы должны быть абстрактными, а какие обычными.
- Механизм ввода исходных данных на усмотрение разработчика.