Inheritance model

Ridi Ferdiana | ridi@acm.org
Version 1.1.0

Overview

- Deriving Classes
- Implementing Methods in Derived Class
- Using Sealed Classes
- Using Interfaces
- Using Abstract Classes

Deriving class

Extending Base Classes

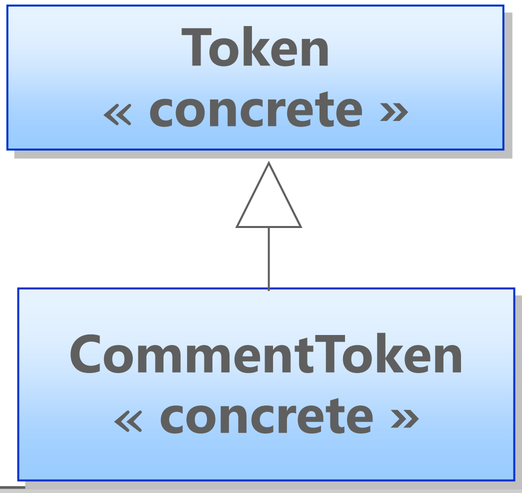
- Syntax for deriving a class from a base class

```
class Token
{
    ...
}
class CommentToken: Token
{
    ...
}
```

Derived class

Base class

Colon



- A derived class inherits most elements of its base class
- A derived class cannot be more accessible than its base class

Accessing Base Class Members

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

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

- Inherited protected members are implicitly protected in the derived class
- Methods of a derived class can access only their inherited protected members
- Protected access modifiers cannot be used in a struct

Calling Base Class Constructors

Constructor declarations must use the base keyword

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

A private base class constructor cannot be accessed by a derived class

Use the base keyword to qualify identifier scope

Implementing Method

Defining Virtual Methods

Syntax: Declare as virtual

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

Virtual methods are polymorphic

Working with Virtual Methods

To use virtual methods:

- You cannot declare virtual methods as static
- You cannot declare virtual methods as private

Overriding Methods

Syntax: Use the override keyword

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

Working with Override Methods

- You can only override identical inherited virtual methods

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

- You must match an override method with its associated virtual method
- You can override an override method
- You cannot explicitly declare an override method as virtual
- You cannot declare an override method as static or private

Using new to Hide Methods

Syntax: Use the new keyword to hide a method

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

}
```

Working with the new Keyword

- Hide both virtual and non-virtual methods

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

- Resolve name clashes in code
- Hide methods that have identical signatures

Practice: Implementing Methods

```
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();
   }
}
```

Quiz: Spot the Bugs

```
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( ) { ... }
}
```

Using Sealed Classes

- You cannot derive from a sealed class
- You can use sealed classes for optimizing operations at run time
- Many .NET Framework classes are sealed: String, StringBuilder, and so on
- Syntax: Use the sealed keyword

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

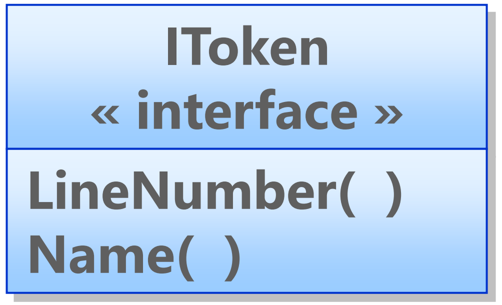
Interface

Declaring Interfaces

Syntax: Use the interface keyword to declare methods

Interface names should begin with a capital "I"

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



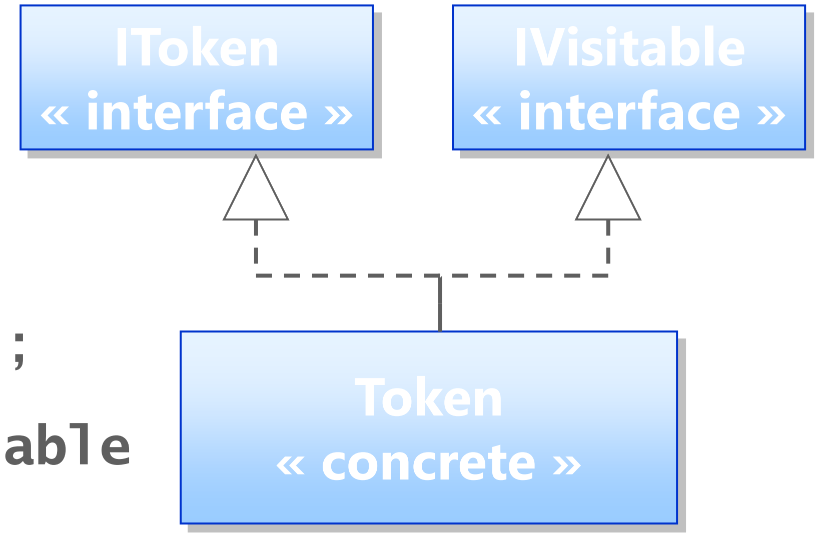
No access specifiers

No method bodies

Implementing Multiple Interfaces

A class can implement zero or more interfaces

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



An interface can extend zero or more interfaces
A class can be more accessible than its base interfaces

An interface cannot be more accessible than its base interfaces
A class must implement all inherited interface methods

Implementing Interface Methods

The implementing method must be the same as the interface method

The implementing method can be virtual or non-virtual

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

Same access
Same return type
Same name
Same parameters

Implementing Interface Methods Explicitly

Use the fully qualified interface method name

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

Restrictions of explicit interface method implementation

- You can only access methods through the interface
- You cannot declare methods as virtual
- You cannot specify an access modifier

Quiz: Spot the Bugs

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

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

Abstract Class

Declaring Abstract Classes

Use the abstract keyword

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

Token
{ abstract }

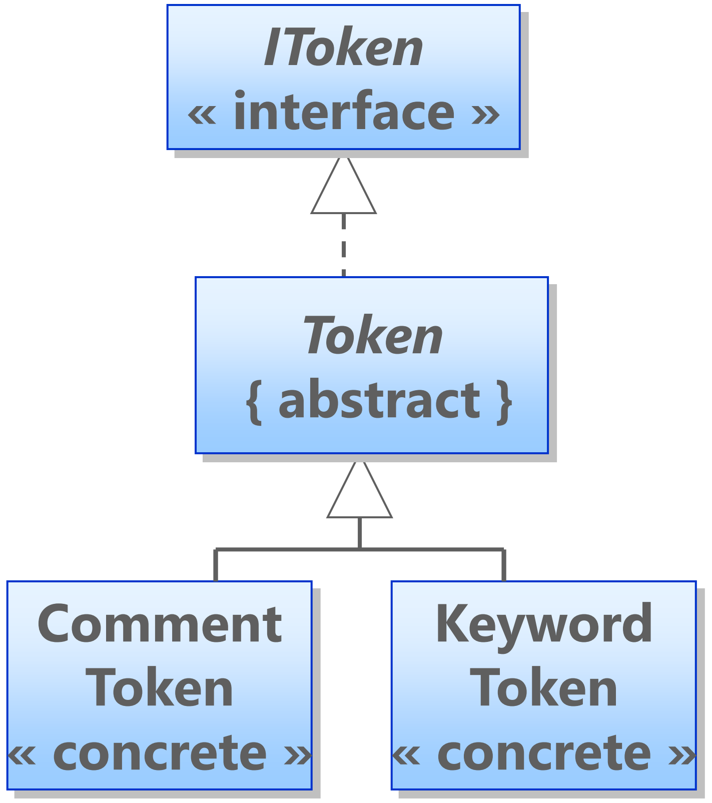
An abstract class cannot be instantiated

Using Abstract Classes in a Class Hierarchy

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

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

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



Comparing Abstract Classes to Interfaces

Similarities

- Neither can be instantiated
- Neither can be sealed

Differences

- Interfaces cannot contain any implementation
- Interfaces cannot declare non-public members
- Interfaces cannot extend non-interfaces

Implementing Abstract Methods

Syntax: Use the abstract keyword

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

Only abstract classes can declare abstract methods

Abstract methods cannot contain a method body

Working with Abstract Methods

Abstract methods are virtual

Override methods can override abstract methods in further derived classes

Abstract methods can override base class methods declared as virtual

Abstract methods can override base class methods declared as override

Quiz: Spot the Bugs

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

1

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

2

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

}
```

3

Review

- Deriving Classes
- Implementing Methods
- Using Sealed Classes
- Using Interfaces
- Using Abstract Classes