Три основных принципа (механизма) ООП

А вот и они:

- Инкапсуляция
- Наследование
- Полиморфизм

Инкапсуляция

Инкапсуляция (англ. ***Encapsulation***) – от словосочетания *in capsulo – в оболочке*

Инкапсуляция – это принцип, согласно которому любой класс, и в более широком смысле – любая часть системы должны рассматриваться как «черный ящик»: пользователь класса или подсистемы должен видеть только интерфейс (т.е. список декларируемых свойств и методов) и не вникать во внутреннюю реализацию

Инкапсуляция

Позволяет минимизировать число связей между классами и подсистемами и, соответственно, упростить независимую реализацию и модификацию классов и подсистем

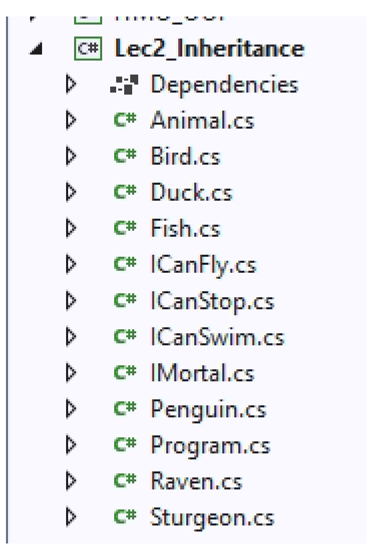
Наследование

Наследование — это возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка (суперкласса), добавляя при необходимости новые свойства и
методы

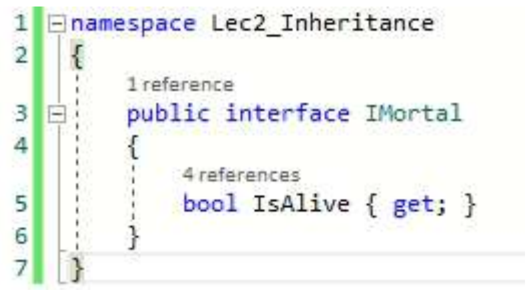
Наследование – очень сильная связь между двумя классами

Наследование помогает переиспользовать код и неразрывно связано с понятием полиморфизма

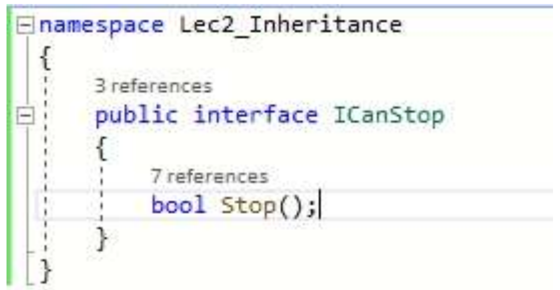
Наследование в C#



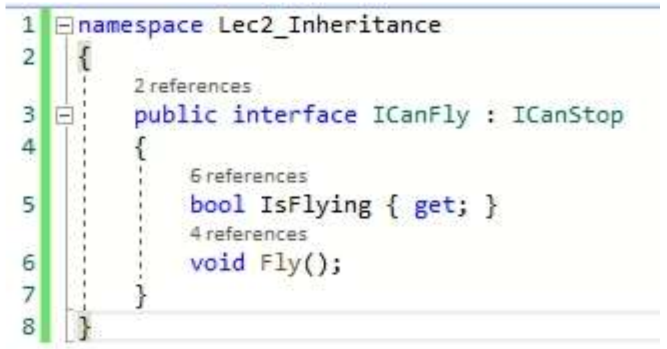
```
Lec2_Inheritance
    Dependencies
    Animal.cs
    Bird.cs
    Duck.cs
    Fish.cs
    ICanFly.cs
    ICanStop.cs
    ICanSwim.cs
    IMortal.cs
    Penguin.cs
    Program.cs
    Raven.cs
    Sturgeon.cs
```



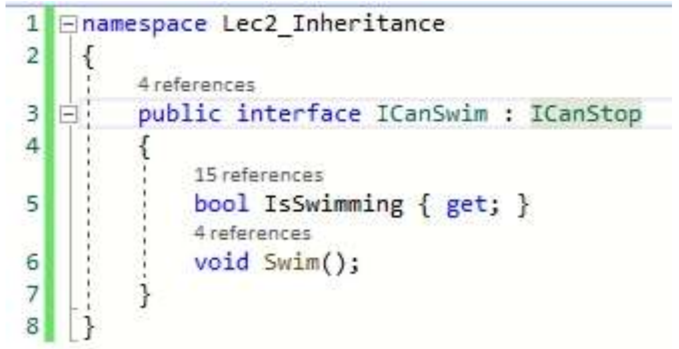
```
namespace Lec2_Inheritance
{
    1 reference
    public interface IMortal
    {
        4 references
        bool IsAlive { get; }
    }
}
```



```
namespace Lec2_Inheritance
{
    3 references
    public interface ICanStop
    {
        7 references
        bool Stop();
    }
}
```

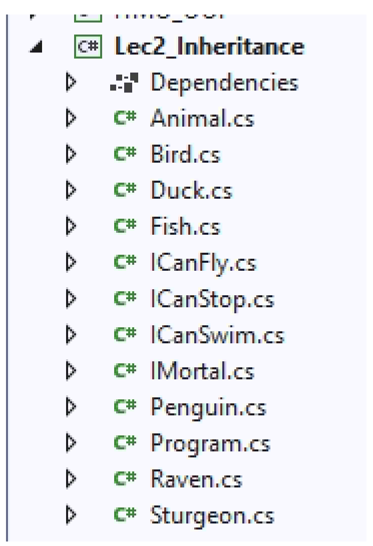


```
namespace Lec2_Inheritance
{
    2 references
    public interface ICanFly : ICanStop
    {
        6 references
        bool IsFlying { get; }
        4 references
        void Fly();
    }
}
```

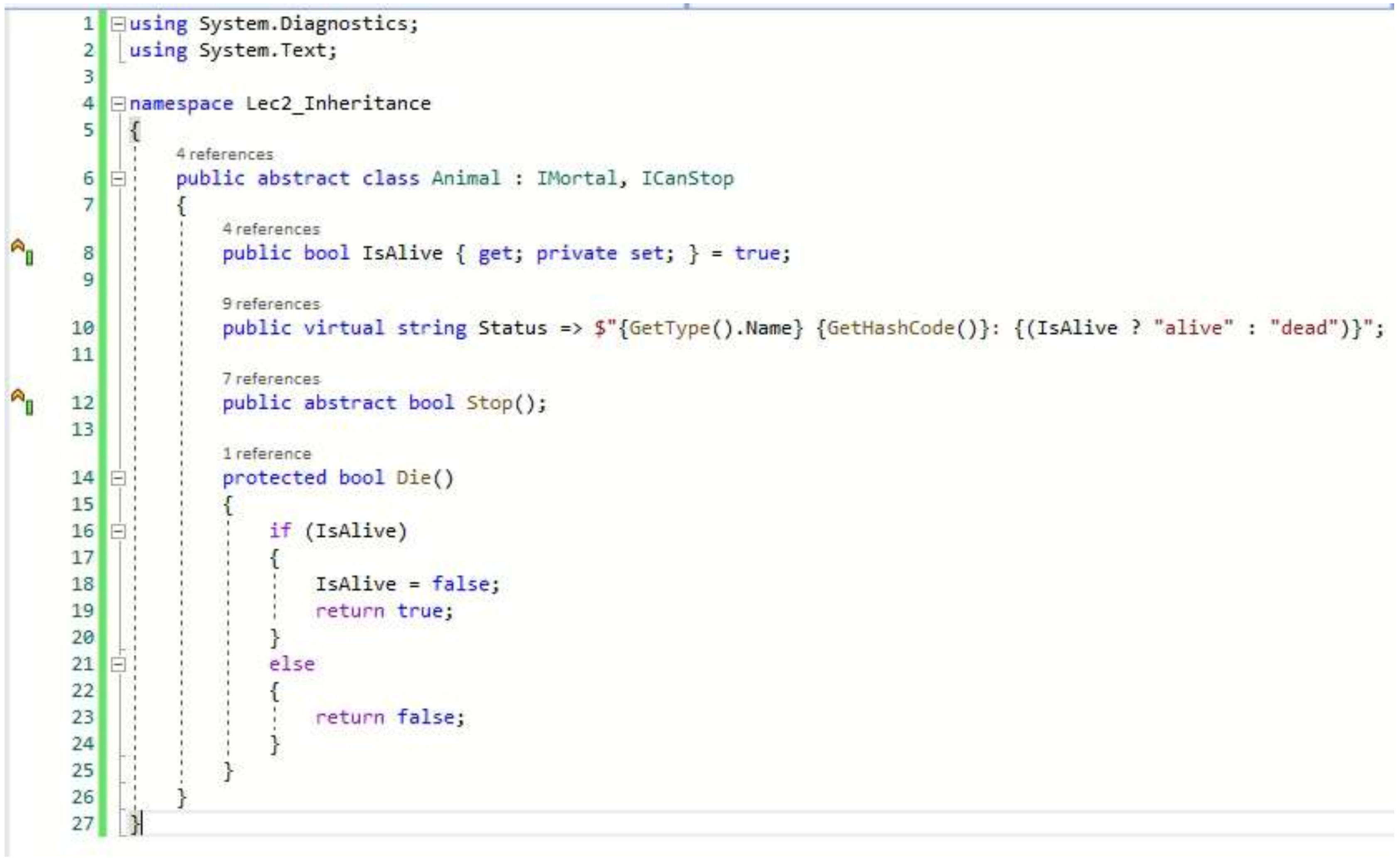


```
namespace Lec2_Inheritance
{
    4 references
    public interface ICanSwim : ICanStop
    {
        15 references
        bool IsSwimming { get; }
        4 references
        void Swim();
    }
}
```

Наследование в C#

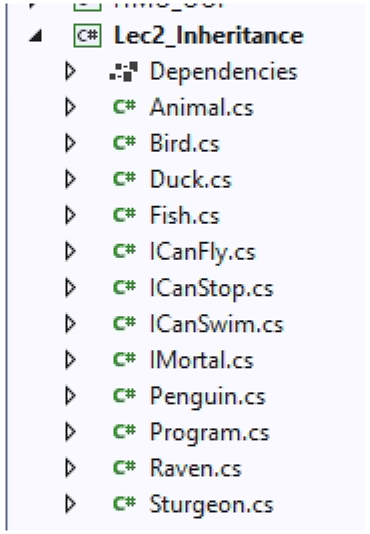


```
Lec2_Inheritance
    Dependencies
    Animal.cs
    Bird.cs
    Duck.cs
    Fish.cs
    ICanFly.cs
    ICanStop.cs
    ICanSwim.cs
    IMortal.cs
    Penguin.cs
    Program.cs
    Raven.cs
    Sturgeon.cs
```

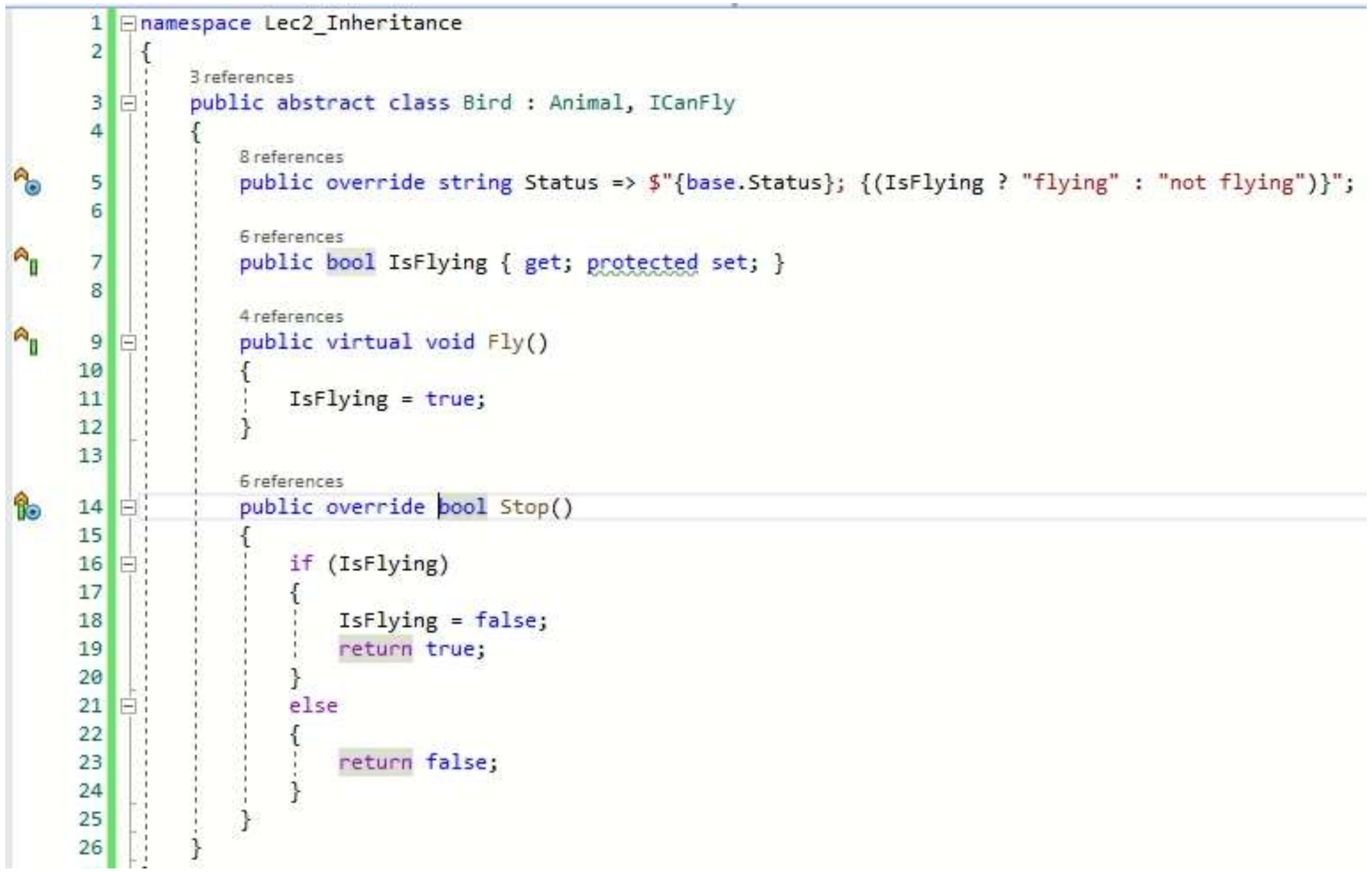


```
using System.Diagnostics;
using System.Text;

namespace Lec2_Inheritance
{
    4 references
    public abstract class Animal : IMortal, ICanStop
    {
        4 references
        public bool IsAlive { get; private set; } = true;

        9 references
        public virtual string Status => $"{GetType().Name} {GetHashCode()}: {(IsAlive ? "alive" : "dead")}";

        7 references
        public abstract bool Stop();

        1 reference
        protected bool Die()
        {
            if (IsAlive)
            {
                IsAlive = false;
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Наследование в C#

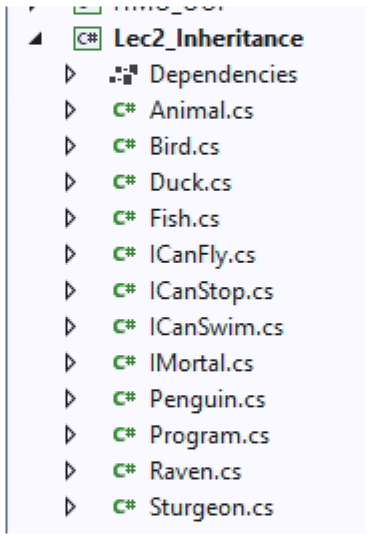


```
◢ Lec2_Inheritance
    ▷ Dependencies
    ▷ Animal.cs
    ▷ Bird.cs
    ▷ Duck.cs
    ▷ Fish.cs
    ▷ ICanFly.cs
    ▷ ICanStop.cs
    ▷ ICanSwim.cs
    ▷ IMortal.cs
    ▷ Penguin.cs
    ▷ Program.cs
    ▷ Raven.cs
    ▷ Sturgeon.cs
```

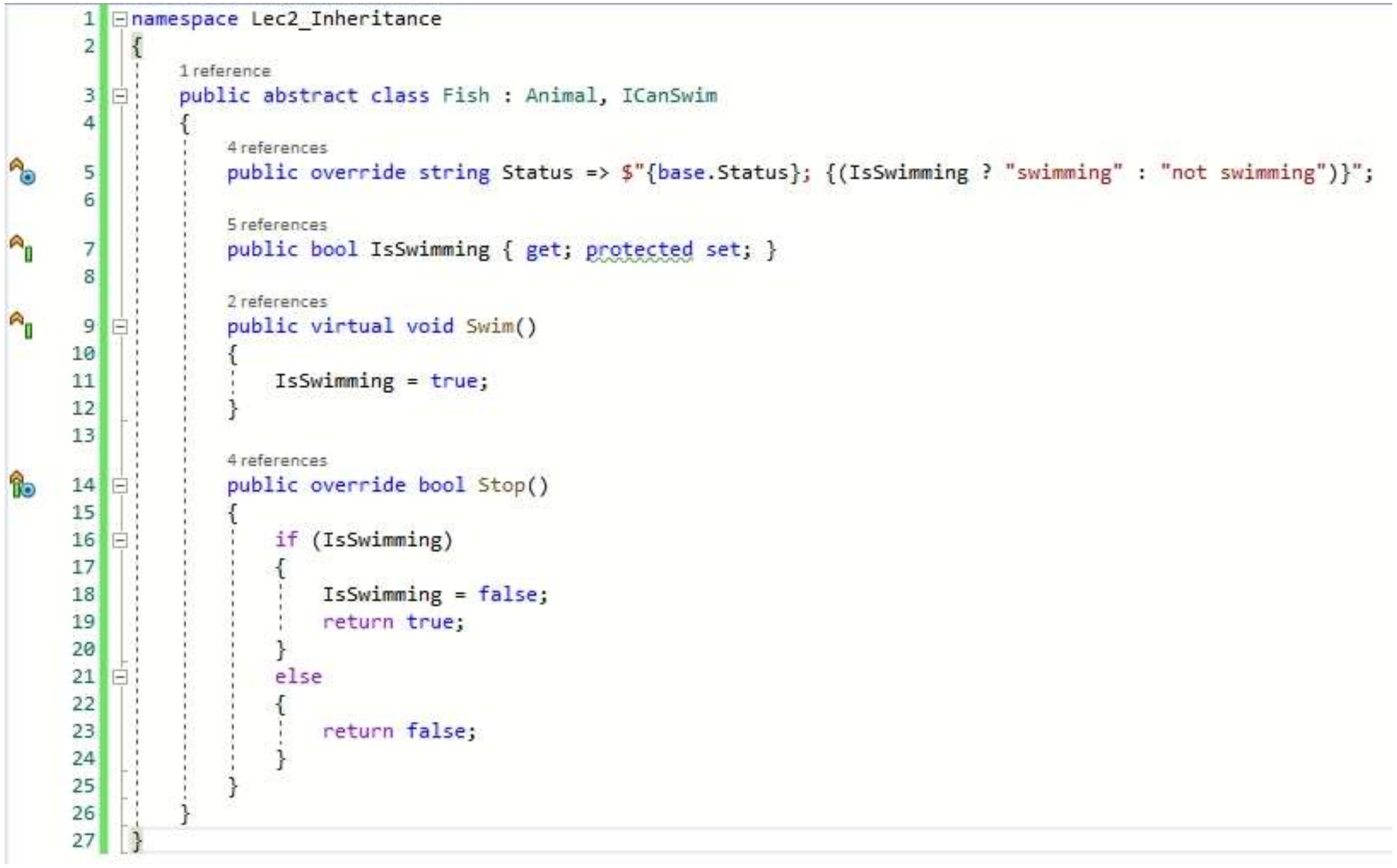


```csharp
namespace Lec2_Inheritance
{
    3 references
    public abstract class Bird : Animal, ICanFly
    {
        8 references
        public override string Status => $"{base.Status}; {(IsFlying ? "flying" : "not flying")}";

        6 references
        public bool IsFlying { get; protected set; }

        4 references
        public virtual void Fly()
        {
            IsFlying = true;
        }

        6 references
        public override bool Stop()
        {
            if (IsFlying)
            {
                IsFlying = false;
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

Наследование в C#

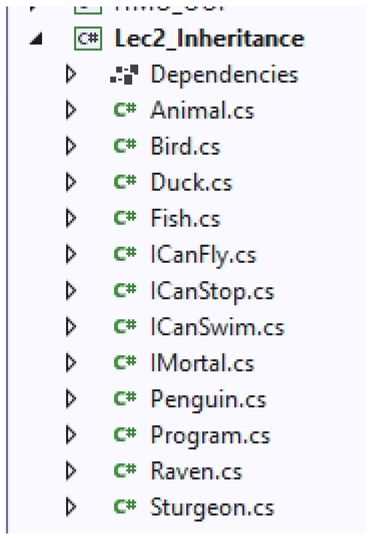


```
Lec2_Inheritance
    Dependencies
    Animal.cs
    Bird.cs
    Duck.cs
    Fish.cs
    ICanFly.cs
    ICanStop.cs
    ICanSwim.cs
    IMortal.cs
    Penguin.cs
    Program.cs
    Raven.cs
    Sturgeon.cs
```

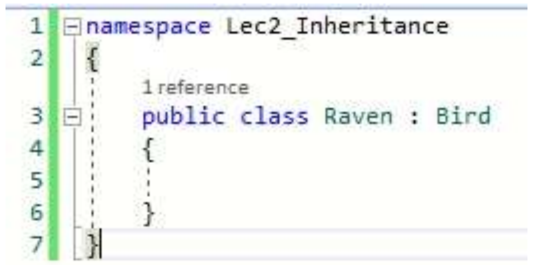


```
namespace Lec2_Inheritance
{
    1 reference
    public abstract class Fish : Animal, ICanSwim
    {
        4 references
        public override string Status => $"{base.Status}; {(IsSwimming ? "swimming" : "not swimming")}";

        5 references
        public bool IsSwimming { get; protected set; }

        2 references
        public virtual void Swim()
        {
            IsSwimming = true;
        }

        4 references
        public override bool Stop()
        {
            if (IsSwimming)
            {
                IsSwimming = false;
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

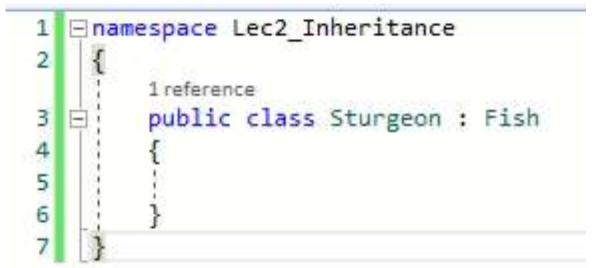
Наследование в C#



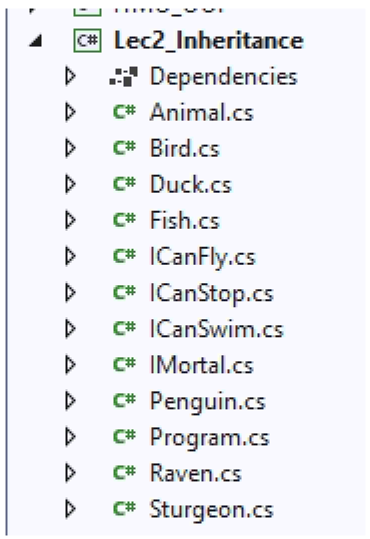
Lec2_Inheritance
- Dependencies
- Animal.cs
- Bird.cs
- Duck.cs
- Fish.cs
- ICanFly.cs
- ICanStop.cs
- ICanSwim.cs
- IMortal.cs
- Penguin.cs
- Program.cs
- Raven.cs
- Sturgeon.cs



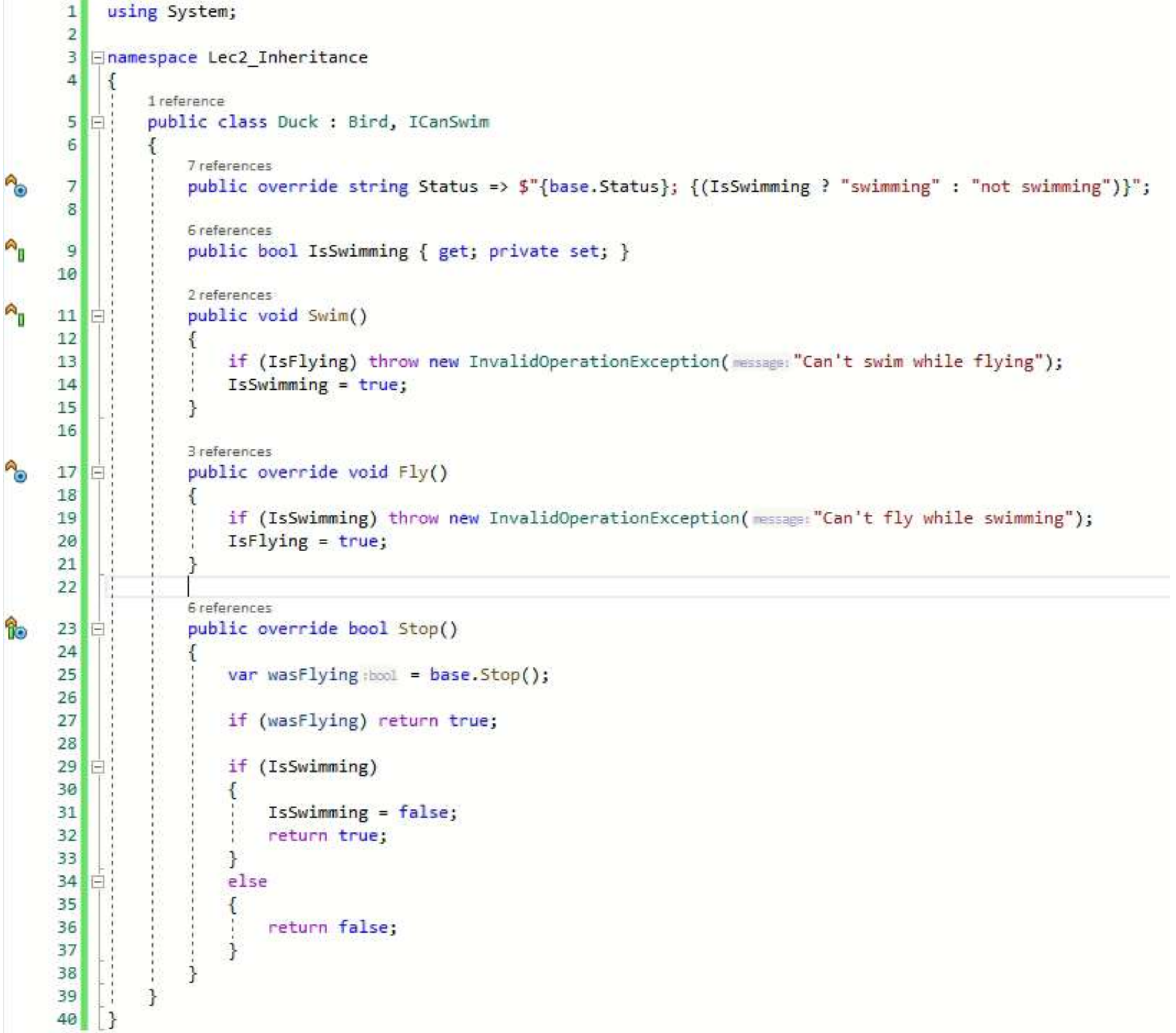
```
namespace Lec2_Inheritance
{
    1 reference
    public class Raven : Bird
    {

    }
}
```



```
namespace Lec2_Inheritance
{
    1 reference
    public class Sturgeon : Fish
    {

    }
}
```

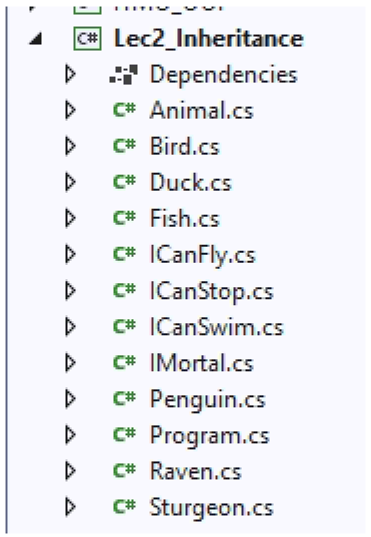


- Lec2_Inheritance
  - Dependencies
  - Animal.cs
  - Bird.cs
  - Duck.cs
  - Fish.cs
  - ICanFly.cs
  - ICanStop.cs
  - ICanSwim.cs
  - IMortal.cs
  - Penguin.cs
  - Program.cs
  - Raven.cs
  - Sturgeon.cs

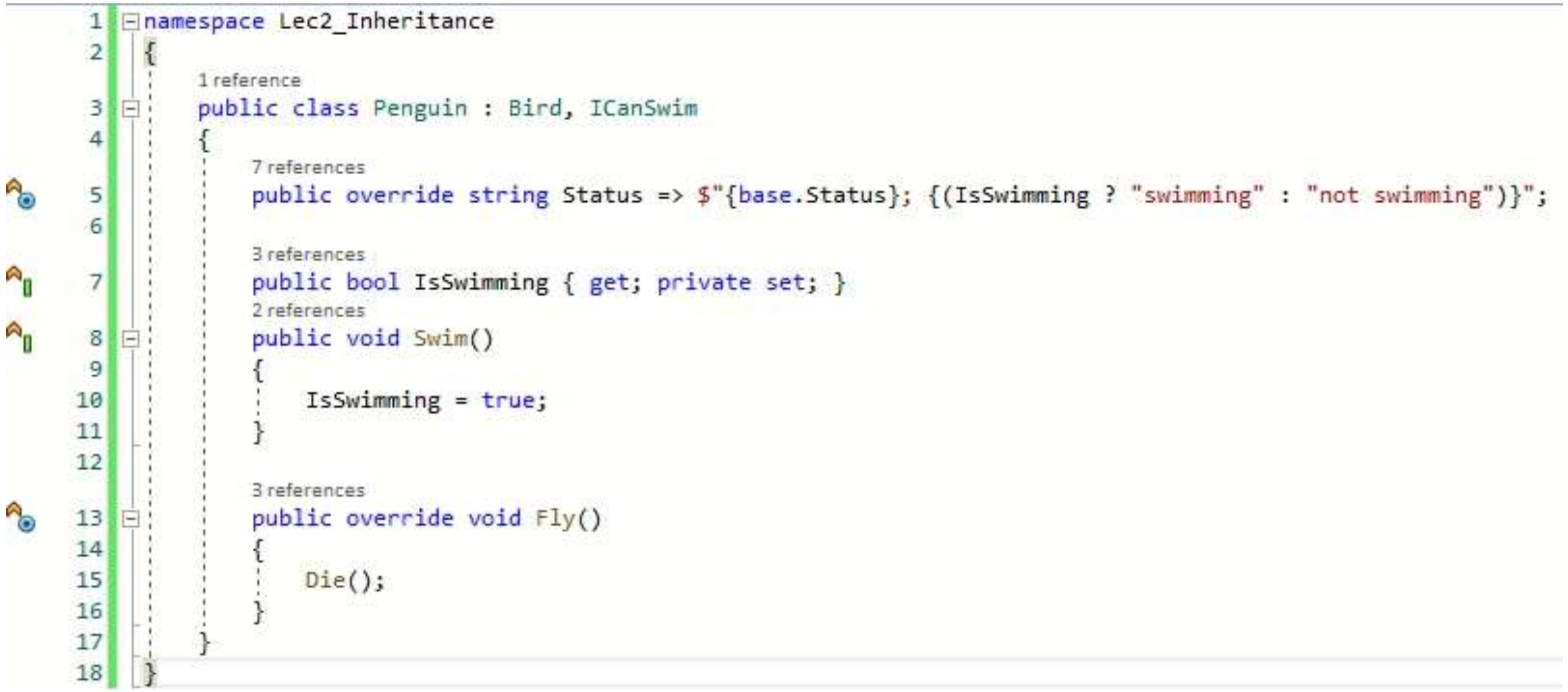


```csharp
using System;

namespace Lec2_Inheritance
{
    public class Duck : Bird, ICanSwim
    {
        public override string Status => $"{base.Status}; {(IsSwimming ? "swimming" : "not swimming")}";

        public bool IsSwimming { get; private set; }

        public void Swim()
        {
            if (IsFlying) throw new InvalidOperationException("Can't swim while flying");
            IsSwimming = true;
        }

        public override void Fly()
        {
            if (IsSwimming) throw new InvalidOperationException("Can't fly while swimming");
            IsFlying = true;
        }

        public override bool Stop()
        {
            var wasFlying = base.Stop();

            if (wasFlying) return true;

            if (IsSwimming)
            {
                IsSwimming = false;
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Наследование в C#

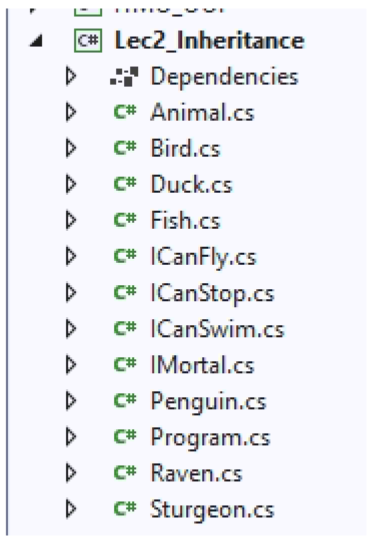


```
Lec2_Inheritance
    Dependencies
    Animal.cs
    Bird.cs
    Duck.cs
    Fish.cs
    ICanFly.cs
    ICanStop.cs
    ICanSwim.cs
    IMortal.cs
    Penguin.cs
    Program.cs
    Raven.cs
    Sturgeon.cs
```

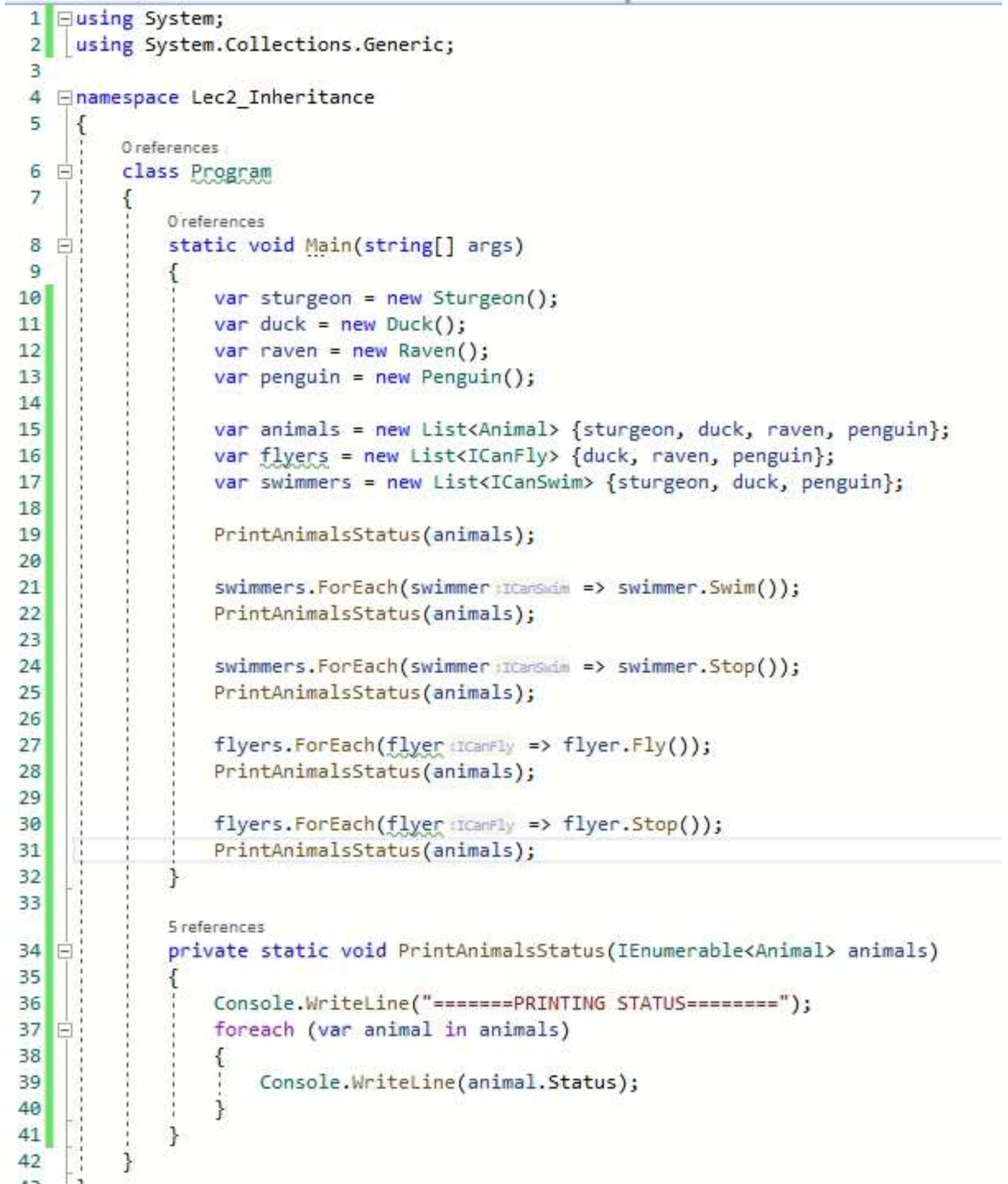


```
namespace Lec2_Inheritance
{
    1 reference
    public class Penguin : Bird, ICanSwim
    {
        7 references
        public override string Status => $"{base.Status}; {(IsSwimming ? "swimming" : "not swimming")}";

        3 references
        public bool IsSwimming { get; private set; }
        2 references
        public void Swim()
        {
            IsSwimming = true;
        }

        3 references
        public override void Fly()
        {
            Die();
        }
    }
}
```

Наследование в C#

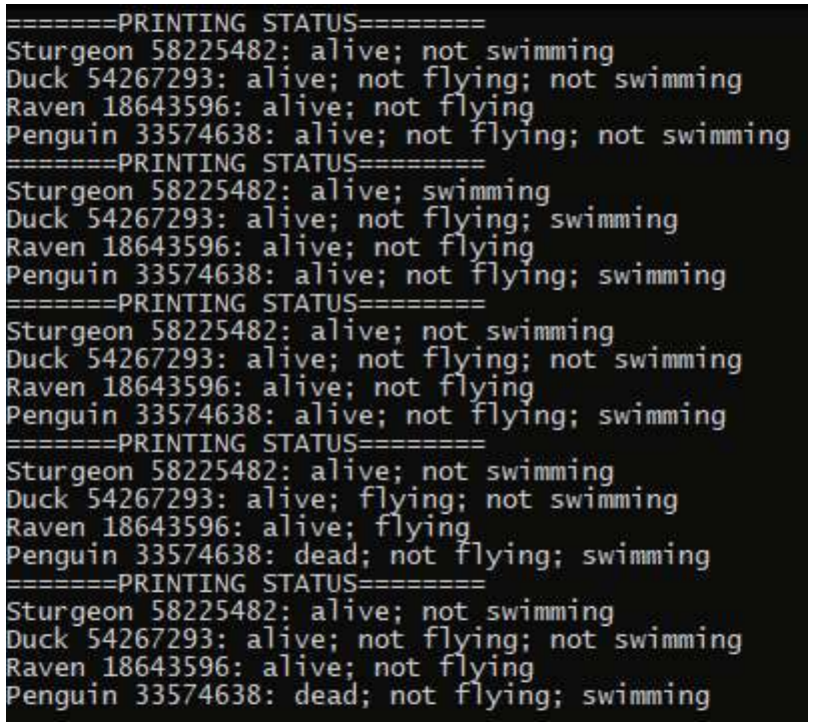


```
Lec2_Inheritance
    Dependencies
    Animal.cs
    Bird.cs
    Duck.cs
    Fish.cs
    ICanFly.cs
    ICanStop.cs
    ICanSwim.cs
    IMortal.cs
    Penguin.cs
    Program.cs
    Raven.cs
    Sturgeon.cs
```



```
using System;
using System.Collections.Generic;

namespace Lec2_Inheritance
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            var sturgeon = new Sturgeon();
            var duck = new Duck();
            var raven = new Raven();
            var penguin = new Penguin();

            var animals = new List<Animal> {sturgeon, duck, raven, penguin};
            var flyers = new List<ICanFly> {duck, raven, penguin};
            var swimmers = new List<ICanSwim> {sturgeon, duck, penguin};

            PrintAnimalsStatus(animals);

            swimmers.ForEach(swimmer :ICanSwim => swimmer.Swim());
            PrintAnimalsStatus(animals);

            swimmers.ForEach(swimmer :ICanSwim => swimmer.Stop());
            PrintAnimalsStatus(animals);

            flyers.ForEach(flyer :ICanFly => flyer.Fly());
            PrintAnimalsStatus(animals);

            flyers.ForEach(flyer :ICanFly => flyer.Stop());
            PrintAnimalsStatus(animals);
        }

        5 references
        private static void PrintAnimalsStatus(IEnumerable<Animal> animals)
        {
            Console.WriteLine("=======PRINTING STATUS========");
            foreach (var animal in animals)
            {
                Console.WriteLine(animal.Status);
            }
        }
    }
```

Наследование в C#



```
=======PRINTING STATUS========
Sturgeon 58225482: alive; not swimming
Duck 54267293: alive; not flying; not swimming
Raven 18643596: alive; not flying
Penguin 33574638: alive; not flying; not swimming
=======PRINTING STATUS========
Sturgeon 58225482: alive; swimming
Duck 54267293: alive; not flying; swimming
Raven 18643596: alive; not flying
Penguin 33574638: alive; not flying; swimming
=======PRINTING STATUS========
Sturgeon 58225482: alive; not swimming
Duck 54267293: alive; not flying; not swimming
Raven 18643596: alive; not flying
Penguin 33574638: alive; not flying; swimming
=======PRINTING STATUS========
Sturgeon 58225482: alive; not swimming
Duck 54267293: alive; flying; not swimming
Raven 18643596: alive; flying
Penguin 33574638: dead; not flying; swimming
=======PRINTING STATUS========
Sturgeon 58225482: alive; not swimming
Duck 54267293: alive; not flying; not swimming
Raven 18643596: alive; not flying
Penguin 33574638: dead; not flying; swimming
```

Полиморфизм

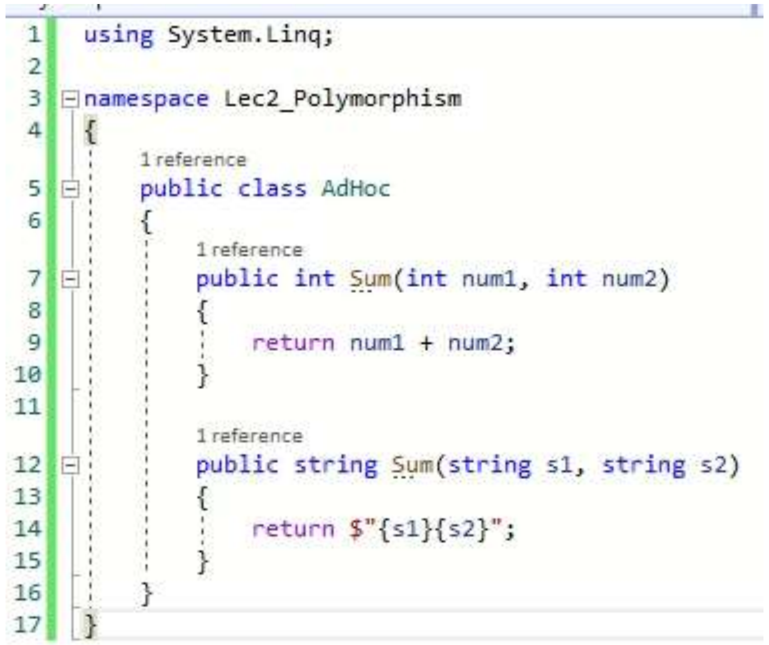
Полиморфизм (англ. ***Polymorphism***) – от греческих слов *poly* – много, *morph* - форма

Полиморфизм — это возможность использовать классы – потомки в контексте, который был предназначен для класса – предка

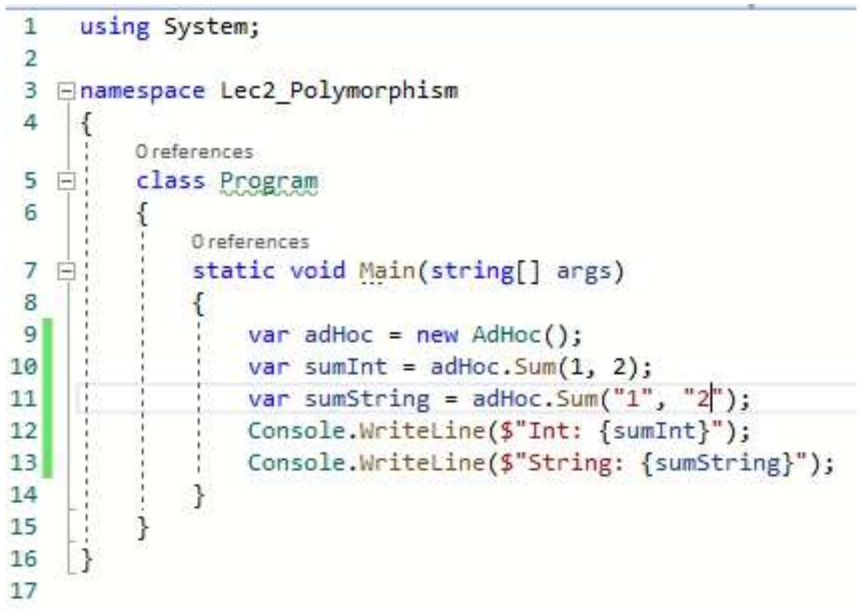
Полиморфизм

- Ad-hoc полиморфизм (статический полиморфизм, раннее связывание)
- Параметрический полиморфизм
- Полиморфизм подтипов (динамический полиморфизм, позднее связывание)

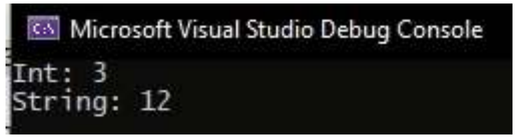
Ad-hoc полиморфизм



```
using System.Linq;

namespace Lec2_Polymorphism
{
    1 reference
    public class AdHoc
    {
        1 reference
        public int Sum(int num1, int num2)
        {
            return num1 + num2;
        }

        1 reference
        public string Sum(string s1, string s2)
        {
            return $"{s1}{s2}";
        }
    }
}
```



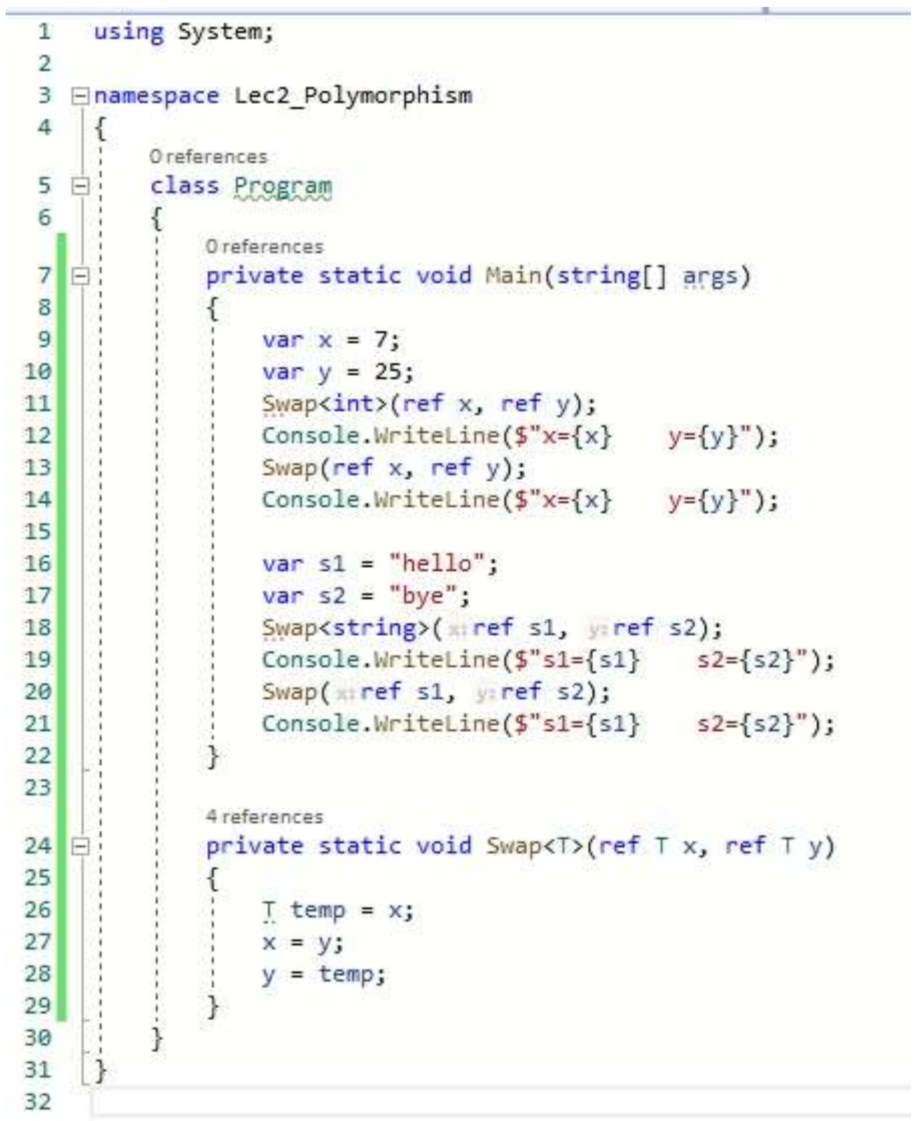
```
using System;

namespace Lec2_Polymorphism
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            var adHoc = new AdHoc();
            var sumInt = adHoc.Sum(1, 2);
            var sumString = adHoc.Sum("1", "2");
            Console.WriteLine($"Int: {sumInt}");
            Console.WriteLine($"String: {sumString}");
        }
    }
}
```



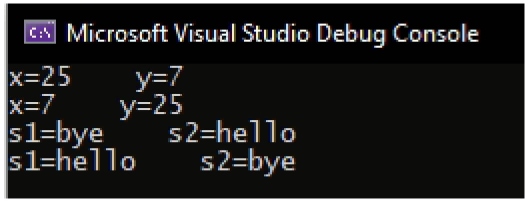
Microsoft Visual Studio Debug Console

```
Int: 3
String: 12
```

Параметрический полиморфизм



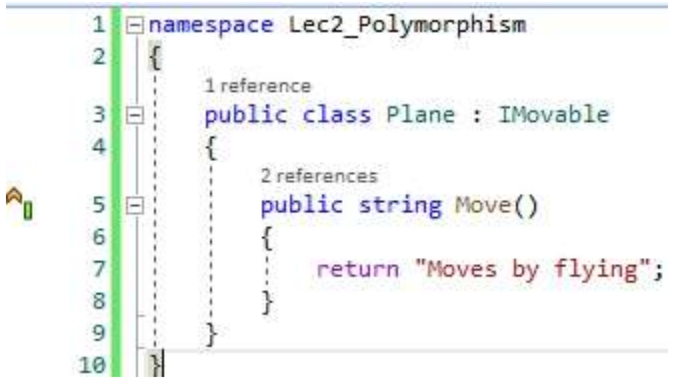
```
using System;

namespace Lec2_Polymorphism
{
    0 references
    class Program
    {
        0 references
        private static void Main(string[] args)
        {
            var x = 7;
            var y = 25;
            Swap<int>(ref x, ref y);
            Console.WriteLine($"x={x}    y={y}");
            Swap(ref x, ref y);
            Console.WriteLine($"x={x}    y={y}");

            var s1 = "hello";
            var s2 = "bye";
            Swap<string>(x: ref s1, y: ref s2);
            Console.WriteLine($"s1={s1}    s2={s2}");
            Swap(x: ref s1, y: ref s2);
            Console.WriteLine($"s1={s1}    s2={s2}");
        }

        4 references
        private static void Swap<T>(ref T x, ref T y)
        {
            T temp = x;
            x = y;
            y = temp;
        }
    }
}
```



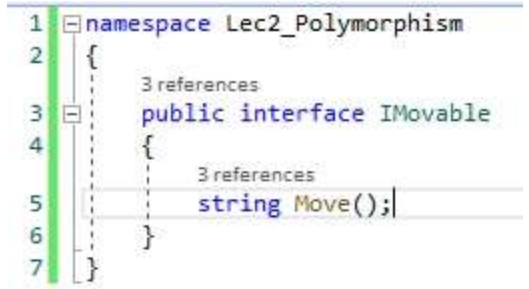
Microsoft Visual Studio Debug Console

```
x=25    y=7
x=7    y=25
s1=bye    s2=hello
s1=hello    s2=bye
```

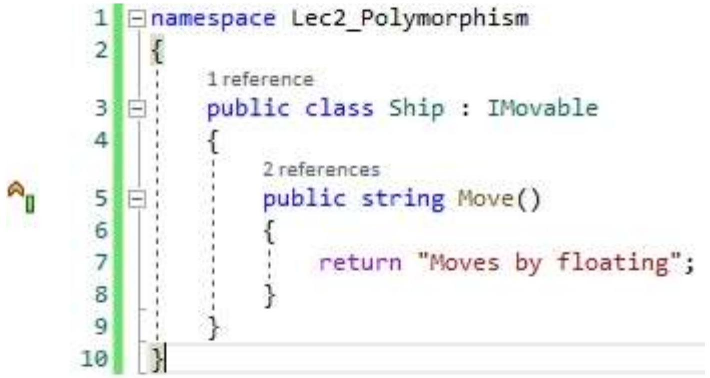
Полиморфизм подтипов



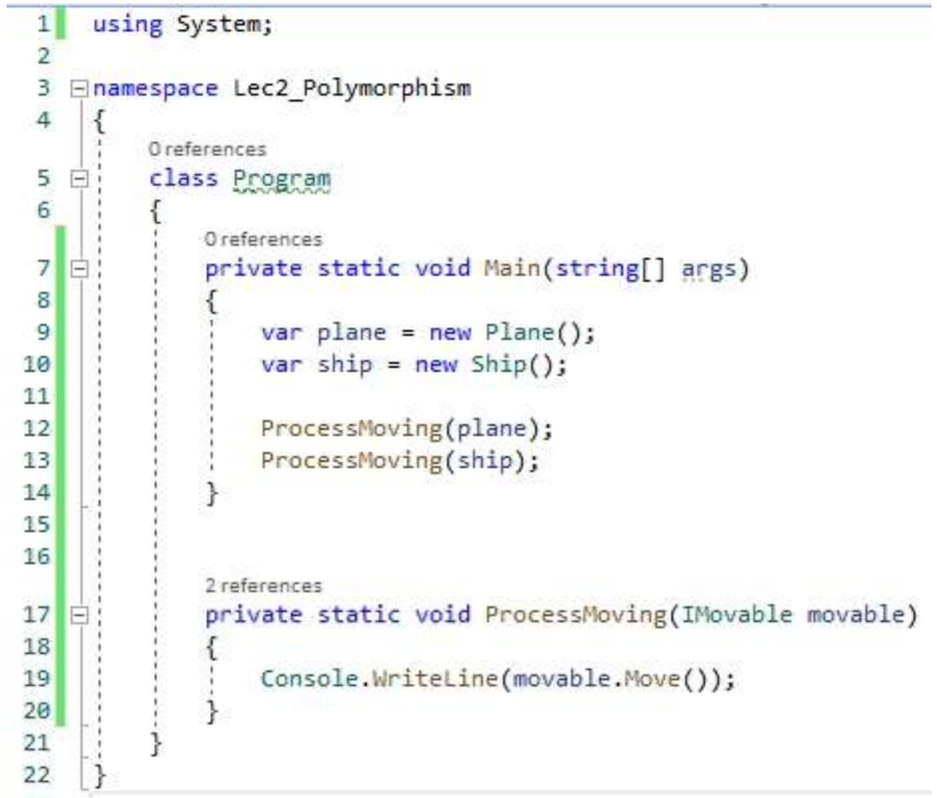
```
namespace Lec2_Polymorphism
{
    1 reference
    public class Plane : IMovable
    {
        2 references
        public string Move()
        {
            return "Moves by flying";
        }
    }
}
```



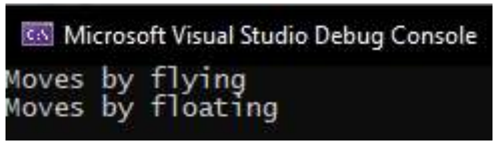
```
namespace Lec2_Polymorphism
{
    3 references
    public interface IMovable
    {
        3 references
        string Move();
    }
}
```



```
namespace Lec2_Polymorphism
{
    1 reference
    public class Ship : IMovable
    {
        2 references
        public string Move()
        {
            return "Moves by floating";
        }
    }
}
```



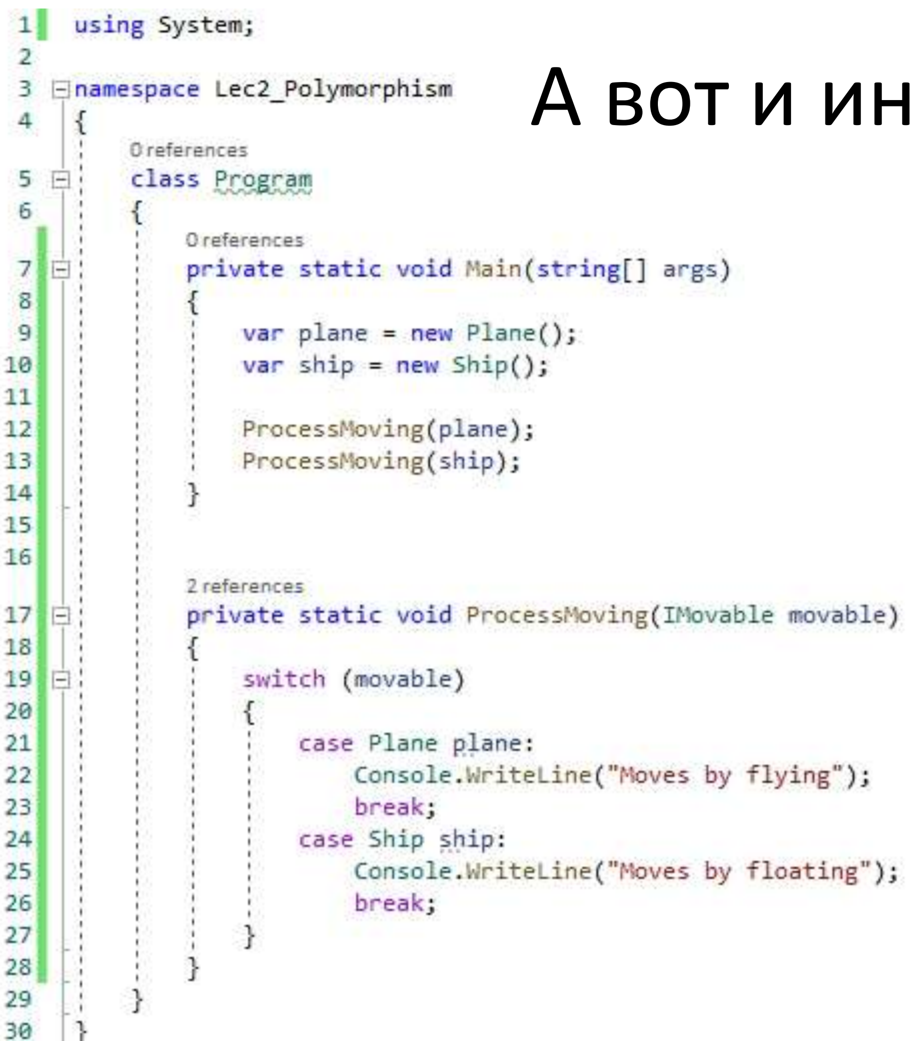
```
using System;

namespace Lec2_Polymorphism
{
    0 references
    class Program
    {
        0 references
        private static void Main(string[] args)
        {
            var plane = new Plane();
            var ship = new Ship();

            ProcessMoving(plane);
            ProcessMoving(ship);
        }


        2 references
        private static void ProcessMoving(IMovable movable)
        {
            Console.WriteLine(movable.Move());
        }
    }
}
```



Microsoft Visual Studio Debug Console

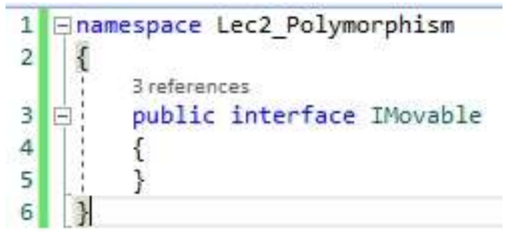
```
Moves by flying
Moves by floating
```

Полиморфизм подтипов. Как НЕ НАДО

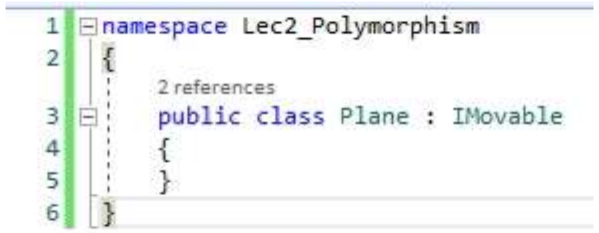


```
using System;

namespace Lec2_Polymorphism
{
    0 references
    class Program
    {
        0 references
        private static void Main(string[] args)
        {
            var plane = new Plane();
            var ship = new Ship();

            ProcessMoving(plane);
            ProcessMoving(ship);
        }


        2 references
        private static void ProcessMoving(IMovable movable)
        {
            switch (movable)
            {
                case Plane plane:
                    Console.WriteLine("Moves by flying");
                    break;
                case Ship ship:
                    Console.WriteLine("Moves by floating");
                    break;
            }
        }
    }
}
```

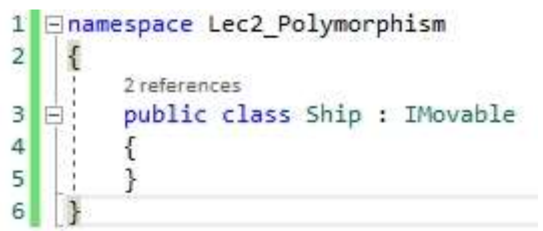
А вот и инкапсуляция нарушена…



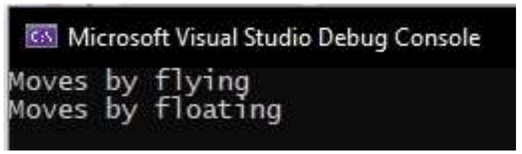
```
namespace Lec2_Polymorphism
{
    3 references
    public interface IMovable
    {
    }
}
```



```
namespace Lec2_Polymorphism
{
    2 references
    public class Plane : IMovable
    {
    }
}
```



```
namespace Lec2_Polymorphism
{
    2 references
    public class Ship : IMovable
    {
    }
}
```



Microsoft Visual Studio Debug Console

```
Moves by flying
Moves by floating
```

Спасибо за внимание!