```
class AHero    { virtual public void Say(){Console.WriteLine("Hi"); } }
```

AHero (Say())

AHomo (Say())

AAnimal

AAss {Say()} ia-ia

ACat {Say()} miau

ADog {Say()} gav-gav

ACock {Say()} ku-ka-re-ku

```
class AAnimal : AHero {   }

class AHomo : AHero
{
 String name;
 String job;
 public AHomo(String name, String job)
   { this.name = name; this.job = job; }
 override public void Say()
  { Console.WriteLine("Hello"); }
}
```

Die Gebruder Grimm

```
ACat says miau-miau
ADog says gav-gav
AAss says Ii-aa
ACock says ku-ka-re-ku
AHomo says Hello
```

```
ACat says gav-gav
ADog says Ii-aa
AAss says ku-ka-re-ku
ACock says Hello
AHomo says miau-miau
```

```
class AAss : AAnimal {
        override public void Say() { Console.WriteLine("Ii-aa"); }
    }

            class ACat : AAnimal
            {
            override public void Say() {  Console.WriteLine("miau-miau"); }
            }

                    class ADog : AAnimal
                        {
                            override public void Say() { Console.WriteLine("gav-gav"); }
                        }

                            class ACock : AAnimal
                             { override public void Say() { Console.WriteLine("ku-ka-re-ku"); }  }
```

```
class ADieBremenStadtMusicanten{
static void Main(string[] args){
   ACat cat = new ACat();
   AAss ass = new AAss();
   ADog dog = new ADog();
   ACock ck=new ACock();
   AHomo h = new AHomo("1","Trompeter");
   AHero[] ah = new AHero[5];
```

```
ah[0] = cat;
ah[1] = dog;
ah[2] = ass;
ah[3]=  ck;
ah[4] = h;
int j;
int k = 0;
```

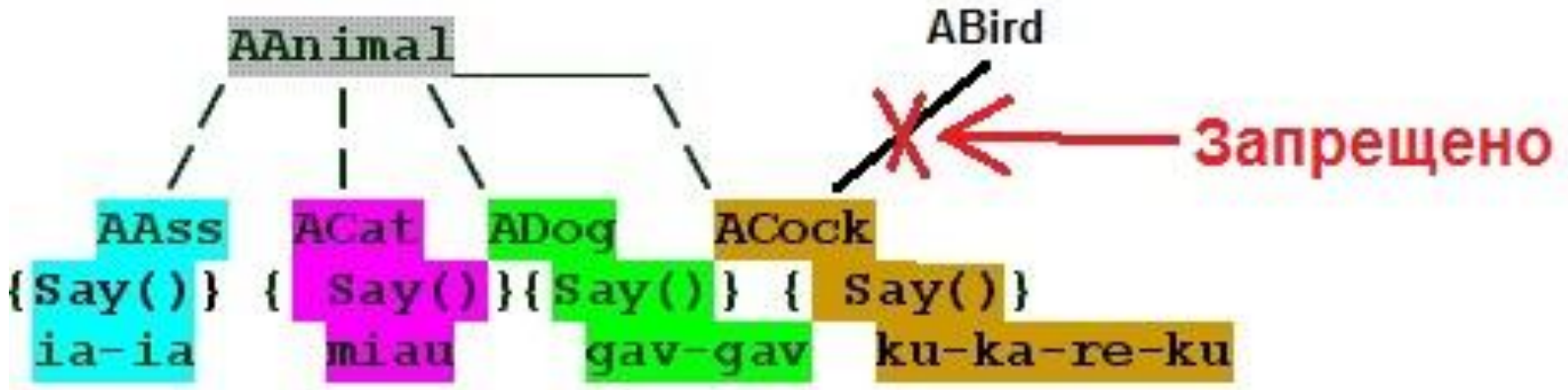
```
for (int i = 0; i < 5; i++)  {
Console.Write(" "+ah[i].GetType().Name+ " says ");
   ah[i].Say(); }
Console.BackgroundColor = ConsoleColor.Blue;
while (k<5) {
Console.Write("  "+ah[k].GetType().Name+" says ");
      j = k++ < 4 ? k : 0;
     ah[j].Say();      }
```

- полиморфизм
- virtual
- override

Интерфейсы

В языке C# запрещено множественное наследование классов.

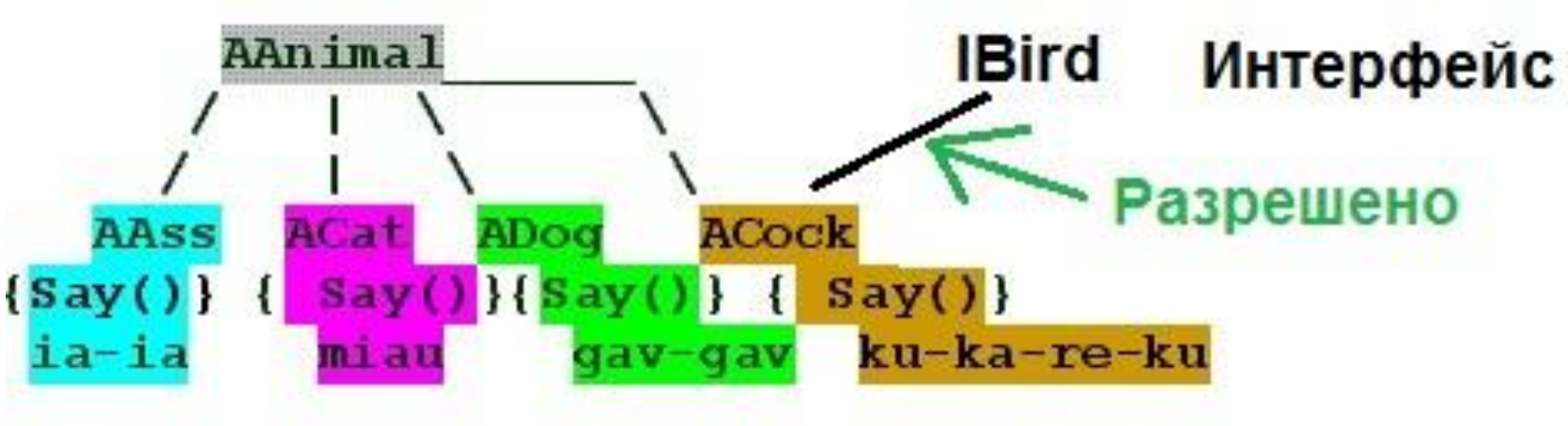
```
class WontWork
  : BaseClassOne, BaseClassTwo
{}
```


AAnimal
ABird
Запрещено
AAss
{Say()}
ia-ia
ACat
{ Say()}
miau
ADog
{Say()}
gav-gav
ACock
{ Say()}
ku-ka-re-ku

Интерфейс представляет собой *набор объявлений* свойств, индексаторов, методов и событий.

```
public interface IBird
{
  void Fly();                    // метод
  double Speed { get; set; }     // свойство
}
```

Класс или структура могут *реализовывать* интерфейс.



В этом случае они берут на себя обязанность предоставить полную реализацию элементов интерфейса (хотя бы пустыми методами).

```
public class ACock : AAnimal,IBird
{
// реализация интерфейса IBird
public void Fly() { Console.WriteLine(“Cock can fly a little");
}
```

Для объявления интерфейса используется ключевое слово interface.

```
public interface IBird
```

Интерфейс содержит только заголовки методов, свойств и событий.

```
void Fly();                                    //
  метод
double Speed { get; set; }                     //
  свойство
```

При объявлении элементов интерфейса не могут использоваться следующие модификаторы:

```
abstract,
public, protected, internal,
private,
 virtual, override,
 static
```

Считается, что все элементы интерфейса имеют public-уровень доступа

```
public interface IBird
```

- Элементы интерфейса допускают явную и неявную реализацию. При *неявной реализации* тип должен содержать открытые экземплярные элементы, имена и сигнатура которых соответствуют элементам интерфейса.

```
public class ACock : AAnimal,IBird
{
// реализация интерфейса IBird
public void Fly()
{
Console.WriteLine("Cock can fly a
  little bit");
public double Speed { get; set; }
}
```

При *явной реализации* элемент типа называется по форме

<*имя интерфейса*>.<*имя элемента*>,

а указание любых модификаторов для элемента при этом запрещается.

```
void IBird.Fly()
{
Console.WriteLine(“Cocks
  flies"); }
double IBird.Speed
{ get; set; }
}
```

Интерфейс IDisposable

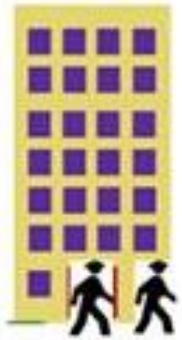
- Интерфейс IDisposable
  public interface IDisposable { Dispose();}
- Позволяет очистить ресурсы в тот момент, когда объект становится ненужным.

Интерфейс IDisposable

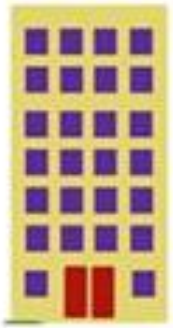
Простая програмка, которая демонстрирует использование метода Dispose может выглядеть так

```
class AStudent : IDisposable
    {
        ArrayList coll =
new ArrayList();
        public void Dispose()
        {
```

Constructor



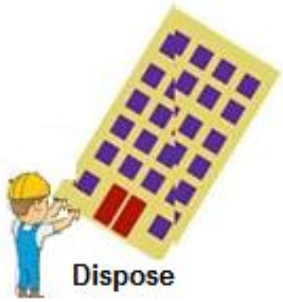
Object being Used



Object unused

Garbage Collector

Object destroyed




Если встретишь
Джавдета
(is IDisposable)
— не трогай его,
он мой

```
class AStudent : IDisposable
{  ArrayList coll = new ArrayList();
public void Dispose()
{ Console.WriteLine("Disposing");
coll.RemoveRange(0,coll.Count);
GC.SuppressFinalize(this);   }
}
class Program
    { static void Main(string[] args)
        {AStudent s = new AStudent();
            if (s is IDisposable) s.Dispose();
        }
    }
```

Свойства

Свойства класса призваны предоставить защищённый доступ к полям.

```
<модификаторы><тип свойства>
  <имя свойства>
{
get {<блоккода>}
set {<блоккода>}
}
```

```
public class AStudent
{
private int _age;
public int Age
{
get  {return _age; }
set
 {
  _age =
       value< 0 ? 0:value;
   // проверка
корректности
 }
}
```

```
AStudent as=
    new AStudent();
ss.Age=22;
int age=as.Age;
```

```
public class AStudent
{
private string _name;
public string Name
{
get  {return _ name; }
set
 {
  _ name = value;
  }
}
```

```
AStudent as=
   new AStudent();
as.Name="Ira";
String name=
              as.Age;
```

```
public class ATimePeriod
{
   privates double sec;
  public string Hour
   {
    get {return sec/3600;};
    set {sec=value*3600;};
   }
}
```

Как правило, свойства открыты, то есть снабжаются модификатором доступа public. Однако иногда логика класса требует разделения права доступа чтения и записи свойства. В C# разрешено при описании свойства указывать модификаторы доступа для аксессоров и мутаторов.

```
public class ASomeClass
{
public int Property
{
get { return 0; }
private set { }
    }
}
```

Достаточно часто свойство содержит только простейший код доступа к полю. Вот фрагмент класса с таким свойством:

```
public class AStudent
{
private string _name;
public string Name
{
get  {return _ name; }
set
 {
  _ name = value;
  }
}
```

Чтобы облегчить описание таких свойств-«обёрток», в C# имеются *автосвойства* (auto properties).

```
public class
            AStudent
{
public string
                Name
{ get;set; }
}
```

В этом случае компилятор сам сгенерирует необходимое поле класса, связанное со свойством.

При необходимости получить аналог классических свойств только для чтения необходимо использовать модификаторы доступа для частей

```
public class AStudent
{
 public string Name
 {
  get;
  private set;
 }
}
```

Индексаторы

При помощи индексаторов осуществляется доступ к коллекции данных, содержащихся в объекте, с использованием привычного синтаксиса для доступа к элементам массива – пары квадратных скобок и индекса.

```
Console.WriteLine(students[i]);
```

```
var students = new AStudents();

students[1] = 8;
students[3] = 4;

  for (int i = 0; i < 5; i++)
  {

  Console.WriteLine(students[i]);
  }
```

```
public class AStudents{
  private int[] _m
                    = new int[5];
  public int this[int i]
  {
     get{ return _m[i] }
                      //students[i]
   set{_m[i] = value; }
                     //students[i]=3;
  }
}
```

```
private bool Belongs(int x, int min, int max)
    {
       return (x >= min) && (x <= max);}
    }
```

```
public class AStudents
   {
   private readonly int[] _m = new int[5];
   public int this[int i]
   {
     get { return Belongs(i, 0, 4) ? _m [i] : 0;}
     set { if (Belongs(i, 0, 4) &&
                  Belongs(value, 1, 10))
                     _m [i] = value; }

}
```

```
for (int i = 0; i < 5; i++)
  {

  Console.WriteLine(students[i]);
  }


foreach (int i in students)
{
Console.WriteLine(i);
}
```

```
class AStudents: IEnumerable, IEnumerator
{
IEnumerator GetEnumerator() {….}
public bool MoveNext()
public void Reset(){}
public object Current{}

}
```

```
    // Реализуем интерейс IEnumerable
public IEnumerator GetEnumerator()
{
     return this;
 }

 // Реализуем интерфейс IEnumerator
 public bool MoveNext()
 {
     if (index == _m.Length - 1)
     {
         Reset();
         return false;
     }

     index++;
     return true;
 }
```

```
public void Reset()
  {
      index = -1;
  }

  public object Current
  {
      get
      {
          return _m[index];
      }
  }
```