第1章　代码无错就是优？——简单工厂模式

1.1　面试受挫

小菜今年计算机专业大四了，学了不少软件开发方面的东西，也学着编了些小程序，踌躇满志，一心要找一个好单位。当投递了无数份简历后，终于收到了一个单位的面试通知，小菜欣喜若狂。

到了人家单位，前台小姐给了他一份题目，上面写着：“请用 C++、Java、C#或 VB.NET 任意一种面向对象语言实现一个计算器控制台程序，要求输入两个数和运算符号，得到结果。”

小菜一看，这个还不简单，三下五除二，10 分钟不到，小菜写完了，感觉也没错误。交卷后，单位说一周内等通知吧。于是小菜只得耐心等待。可是半个月过去了，什么消息也没有，小菜很纳闷，我的代码实现了呀，为什么不给我机会呢。

时间：2 月 26 日 20 点　　**地点：**大鸟房间　　**人物：**小菜、大鸟

小菜找到从事软件开发工作七年的表哥大鸟，请教原因，大鸟问了题目和了解了小菜代码的细节以后，哈哈大笑，说道：“小菜呀小菜，你上当了，人家单位出题的意思，你完全都没明白，当然不会再联系你了。”

小菜说：“我的代码有错吗？单位题目不就是要我实现一个计算器的代码吗，我这样写有什么问题。”

```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("请输入数字A：");
        string A = Console.ReadLine();
        Console.Write("请选择运算符号(+、-、*、/)：");
        string B = Console.ReadLine();
        Console.Write("请输入数字B：");
        string C = Console.ReadLine();
        string D = "";

        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
```

```
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (O == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));

        Console.WriteLine("结果是: " + D);
    }
}
```

1.2 初学者代码毛病

大鸟说:“且先不说出题人的意思，单就你现在的代码，就有很多不足的地方需要改进。”

```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("请输入数字A: ");
        string A = Console.ReadLine();
        Console.Write("请选择运算符号(+、-、*、/): ");
        string B = Console.ReadLine();
        Console.Write("请输入数字B: ");
        string C = Console.ReadLine();
        string D = "";

        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (O == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));
        Console.WriteLine("结果是: " + D);
    }
}
```

这样命名是非常不规范的

判断分支，你这样的写法，意味着每个条件都要做判断，等于计算机做了三次无用功

如果除数时，客户输入了0怎么办，如果用户输入的是字符符号而不是数字怎么办

1.3 代码规范

“哦，说得没错，这个我以前听老师说过，可是从来没有在意过，我马上改，改完再给你看看。”

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("请输入数字 A: ");
            string strNumberA = Console.ReadLine();
            Console.Write("请选择运算符号(+、-、*、/): ");
            string strOperate = Console.ReadLine();
            Console.Write("请输入数字 B: ");
            string strNumberB = Console.ReadLine();
            string strResult = "";
            switch (strOperate)
            {
                case "+":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        + Convert.ToDouble(strNumberB));
                    break;
                case "-":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        - Convert.ToDouble(strNumberB));
                    break;
                case "*":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        * Convert.ToDouble(strNumberB));
                    break;
                case "/":
                    if (strNumberB != "0")
                        strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                            / Convert.ToDouble(strNumberB));
                    else
                        strResult = "除数不能为 0";
                    break;
            }
            Console.WriteLine("结果是: " + strResult);
            Console.ReadLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine("您的输入有错: " + ex.Message);
        }
    }
}
```

大鸟：“吼吼，不错，不错，改得很快嘛？至少就目前代码来说，实现计算器是没有问题了，但这样写出的代码是否合出题人的意思呢？”

小菜：“你的意思是面向对象？”

大鸟：“哈，小菜非小菜也！”

1.4 面向对象编程

小菜：“我明白了，他说用任意一种面向对象语言实现，那意思就是要用面向对象的编程方法去实现，对吗？OK，这个我学过，只不过当时我没想到而已。”

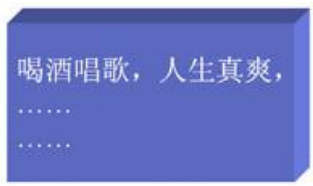
大鸟：“所有编程初学者都会有这样的问题，就是碰到问题就直觉地用计算机能够理解的逻辑来描述和表达待解决的问题及具体的求解过程。这其实是用计算机的方式去思考，比如计算器这个程序，先要求输入两个数和运算符号，然后根据运算符号判断选择如何运算，得到结果，这本身没有错，但这样的思维却使得我们的程序只为满足实现当前的需求，程序不容易维护，不容易扩展，更不容易复用。从而达不到高质量代码的要求。”

小菜：“鸟哥呀，我有点糊涂了，如何才能容易维护，容易扩展，又容易复用呢，能不能具体点？”

1.5 活字印刷，面向对象

大鸟：“这样吧，我给你讲个故事。你就明白了。”

“话说三国时期，曹操带领百万大军攻打东吴，大军在长江赤壁驻扎，军船连成一片，眼看就要灭掉东吴，统一天下，曹操大悦，于是大宴众文武，在酒席间，曹操诗性大发，不觉吟道：‘喝酒唱歌，人生真爽。……’。众文武齐呼：‘丞相好诗！’于是一臣子速命印刷工匠刻版印刷，以便流传天下。”



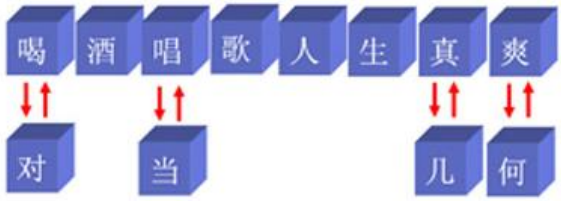
“样张出来给曹操一看，曹操感觉不妥，说道：‘喝与唱，此话过俗，应改为‘对酒当歌’较好！’，于是此臣就命工匠重新来过。工匠眼看连夜刻版之工，彻底白费，心中叫苦不迭。只得照办。”

“样张再次出来请曹操过目，曹操细细一品，觉得还是不好，说：‘人生真爽太过直接，应改问语才够意境，因此应改为‘对酒当歌，人生几何？……’当臣转告工匠之时，工匠晕倒……！”

“小菜你说，这里面问题出在哪里？”大鸟问道。

小菜说：“是不是因为三国时期活字印刷还未发明，所以要改字的时候，就必须要整个刻板全部重新刻。”

大鸟：“说得好！如果是有了活字印刷，则只需更改四个字就可，其余工作都未白做。岂不妙哉。”



“第一，要改，只需更改要改之字，此为**可维护**；第二，这些字并非用完这次就无用，完全可以在后来的印刷中重复使用，此乃**可复用**；第三，此诗若要加字，只需另刻字加入即可，这是**可扩展**；第四，字的排列其实可能是竖排可能是横排，此时只需将活字移动就可做到满足排列需求，此是**灵活性好**。”

“而在活字印刷术出现之前，上面的四种特性都无法满足，要修改，必须重刻，要加字，必须重刻，要重新排列，必须重刻，印完这本书后，此版已无任何可再利用价值。”

小菜：“是的，小时候，我一直奇怪，为何火药、指南针、造纸术都是从无到有，从未知到发现的伟大发明，而活字印刷仅仅是从刻版印刷到活字印刷的一次技术上的进步，为何不是评印刷术为四大发明之一呢？原来活字印刷的成功是这个原因。”

1.6　面向对象的好处

大鸟：“哈，这下你明白了？我以前也不懂，不过做了软件开发几年后，经历了太多的类似曹操这样的客户要改变需求，更改最初想法的事件，才逐渐明白当中的道理。其实客观地说，客户的要求也并不过份，不就是改几个字吗，但面对已完成的程序代码，却是需要几乎重头来过的尴尬，这实在是痛苦不堪。说白了，原因就是因为我们原先所写的程序，不容易维护，灵活性差，不容易扩展，更谈不上复用，因此面对需求变化，加班加点，对程序动大手术的那种无奈也就成了非常正常的事了。之后当我学习了面向对象的分析设计编程思想，开始考虑**通过封装、继承、多态把程序的耦合度降低**，传统印刷术的问题就在于所有的字都刻在同一版面上造成耦合度太高所致，开始**用设计模式使得程序更加的灵活，容易修改，并且易于复用**。体会到面向对象带来的好处，那种感觉应该就如同是一中国酒鬼第一次喝到了茅台，西洋酒鬼第一次喝到了 XO 一样，怎个爽字可形容呀。”

“是呀是呀，你说得没错，中国古代的四大发明，另三种应该都是科技的进步，伟大的创造或发现。而唯有活字印刷，实在是思想的成功，面向对象的胜利。”小菜也兴奋起来：“你的意思是，面试公司出题的目的是要我写出容易维护，容易扩展，又容易复用的计算器程序？那该如何做呀？”

1.7 复制 vs.复用

大鸟：“比如说，我现在要求你再写一个 Windows 的计算器，你现在的代码能不能复用呢？”

小菜：“那还不简单，把代码复制过去不就行了吗？改动又不大，不算麻烦。”

大鸟：“小菜看来还是小菜呀，有人说初级程序员的工作就是 Ctrl+C 和 Ctrl+V，这其实是非常不好的编码习惯，因为当你的代码中重复的代码多到一定程度，维护的时候，可能就是一场灾难。越大的系统，这种方式带来的问题越严重，编程有一原则，就是用尽可能的办法去避免重复。想想看，你写的这段代码，有哪些是和控制台无关的，而只是和计算器有关的？”

小菜：“你的意思是分一个类出来？ 哦，对的，让计算和显示分开。”

1.8 业务的封装

大鸟：“准确地说，就是让业务逻辑与界面逻辑分开，让它们之间的耦合度下降。只有分离开，才可以达到容易维护或扩展。”

小菜：“让我来试试看。”

Operation 运算类

```
public class Operation
{
    public static double GetResult(double numberA, double numberB, string operate)
    {
        double result = 0d;
        switch (operate)
        {
            case "+":
                result = numberA + numberB;
                break;
            case "-":
```

```
                result = numberA - numberB;
                break;
            case "*":
                result = numberA * numberB;
                break;
            case "/":
                result = numberA / numberB;
                break;
        }
        return result;
    }
}
```

客户端代码

```
static void Main(string[] args)
{
    try
    {
        Console.Write("请输入数字 A：");
        string strNumberA = Console.ReadLine();
        Console.Write("请选择运算符号(+、-、*、/)：");
        string strOperate = Console.ReadLine();
        Console.Write("请输入数字 B：");
        string strNumberB = Console.ReadLine();
        string strResult = "";
        strResult = Convert.ToString(Operation.GetResult(Convert.ToDouble(strNumberA),
        Convert.ToDouble(strNumberB), strOperate));
        Console.WriteLine("结果是：" + strResult);
        Console.ReadLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine("您的输入有错：" + ex.Message);
    }
}
```

小菜：“鸟哥，我写好了，你看看！”

大鸟：“孺鸟可教也，写得不错，这样就完全把业务和界面分离了。”

小菜心中暗骂：“你才是鸟呢。”口中说道：“如果你现在要我写一个 Windows 应用程序的计算器，我就可以复用这个运算类（Operation）了。”

大鸟：“不单是 Windows 程序，Web 版程序需要运算可以用它，PDA、手机等需要移动系统的软件需要运算也可以用它呀。”

小菜：“哈，面向对象不过如此。下回写类似代码不怕了。”

大鸟：“别急，仅此而已，实在谈不上完全面向对象，你只用了面向对象三大特性中的一个，还有两个没用呢？”

小菜：“面向对象三大特性不就是封装、继承和多态吗，这里我用到的应该是封装。这还不够吗？我实在看不出，这么小的程序如何用到继承。至于多态，其实我一直也不太了解它到底有什么好处，如何使用它。”

大鸟：“慢慢来，要学的东西多着呢，你好好想想该如何应用面向对象的继承和多态。”

1.9 紧耦合 vs.松耦合

第二天。

小菜问道：“你说计算器这样的小程序还可以用到面向对象三大特性？继承和多态怎么可能用得上，我实在不能理解。”

大鸟：“小菜很有钻研精神嘛，好，今天我让你功力加深一级。你先要考虑一下，你昨天写的这个代码，能否做到很灵活的可修改和扩展呢？”

小菜：“我已经把业务和界面分离了呀，这不是很灵活了吗？”

大鸟：“那我问你，现在如果我希望增加一个开根（sqrt）运算，你如何改？”

小菜：“那只需要改 Operation 类就行了，在 switch 中加一个分支就行了。”

大鸟：“问题是你要加一个平方根运算，却需要让加减乘除的运算都得来参与编译，如果你一不小心，把加法运算改成了减法，这岂不是大大的糟糕。打个比方，如果现在公司要求你为公司的薪资管理系统做维护，原来只有技术人员（月薪），市场销售人员（底薪+提成），经理（年薪+股份）三种运算算法，现在要增加兼职工作人员（时薪）的算法，但按照你昨天的程序写法，公司就必须要把包含原三种算法的运算类给你，让你修改，你如果心中小算盘一打，‘TMD，公司给我的工资这么低，我真是郁闷，这下有机会了’，于是你除了增加了兼职算法以外，在技术人员（月薪）算法中写了一句

```
if (员工是小菜)
{
    salary = salary * 1.1;
}
```

那就意味着，你的月薪每月都会增加 10%（小心被抓去坐牢），本来是让你加一个功能，却使得原有的运行良好的功能代码产生了变化，这个风险太大了。你明白了吗？”

小菜：“哦，你的意思是，我应该把加减乘除等运算分离，修改其中一个不影响另外的几个，增加运算算法也不影响其他代码，是这样吗？”

大鸟：“自己想去吧，如何用继承和多态，你应该有感觉了。”

小菜：“OK，我马上去写。”

Operation 运算类

```
public class Operation
{
    private double _numberA = 0;
    private double _numberB = 0;

    public double NumberA
    {
        get  {  return _numberA;  }
        set  {  _numberA = value;  }
    }
    public double NumberB
    {
        get  {  return _numberB;  }
        set  {  _numberB = value;  }
    }
    public virtual double GetResult()
    {
        double result = 0;
        return result;
    }
}
```

加减乘除类

```
class OperationAdd : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA + NumberB;
        return result;
    }
}

class OperationSub : Operation
{
   public override double GetResult()
    {
        double result = 0;
        result = NumberA - NumberB;
        return result;
    }
```

加法类，继承运算类

减法类，继承运算类

```
}

class OperationMul : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA * NumberB;
        return result;
    }
}

class OperationDiv : Operation
{
    public override double GetResult()
    {
        double result = 0;
        if (NumberB==0)
            throw new Exception("除数不能为0。");
        result = NumberA / NumberB;
        return result;
    }
}
```

乘法类，继承运算类

除法类，继承运算类

小菜：“大鸟哥，我按照你说的方法写出来了一部分，首先是一个运算类，它有两个 Number 属性，主要用于计算器的前后数，然后有一个虚方法 GetResult()，用于得到结果，然后我把加减乘除都写成了运算类的子类，继承它后，重写了 GetResult()方法，这样如果要修改任何一个算法，就不需要提供其他算法的代码了。但问题来了，我如何让计算器知道我是希望用哪一个算法呢？”

(作者注：以上代码读者如果看着吃力，说明您对继承、多态、虚方法、方法重写等概念的理解不够，建议先阅读本书附录一，理解这些基本概念后再继续往下阅读。)

1.10 简单工厂模式

大鸟：“写得很不错嘛，大大超出我的想象了，你现在的问题其实就是如何去实例化对象的问题，哈，今天心情不错，再教你一招‘简单工厂模式’，也就是说，到底要实例化谁，将来会不会增加实例化的对象，比如增加开根运算，这是很容易变化的地方，应该考虑用一个单独的类来做这个创造实例的过程，这就是工厂，来，我们看看这个类如何写。”

简单运算工厂类

```
public class OperationFactory
{
    public static Operation createOperate(string operate)
```

```
    {
        Operation oper = null;
        switch (operate)
        {
            case "+":
                oper = new OperationAdd();
                break;
            case "-":
                oper = new OperationSub();
                break;
            case "*":
                oper = new OperationMul();
                break;
            case "/":
                oper = new OperationDiv();
                break;
        }
        return oper;
    }
}
```

大鸟：“哈，看到了吧，这样子，你只需要输入运算符号，工厂就实例化出合适的对象，通过多态，返回父类的方式实现了计算器的结果。”

客户端代码

```
Operation oper;
oper = OperationFactory.createOperate("+");
oper.NumberA = 1;
oper.NumberB = 2;
double result = oper.GetResult();
```

大鸟：“哈，界面的实现就是这样的代码，不管你是控制台程序，Windows 程序，Web 程序，PDA 或手机程序，都可以用这段代码来实现计算器的功能，如果有一天我们需要更改加法运算，我们只需要改哪里？”

小菜：“改 OperationAdd 就可以了。”

大鸟：“那么我们需要增加各种复杂运算，比如平方根，立方根，自然对数，正弦余弦等，如何做？”

小菜：“只要增加相应的运算子类就可以了呀。”

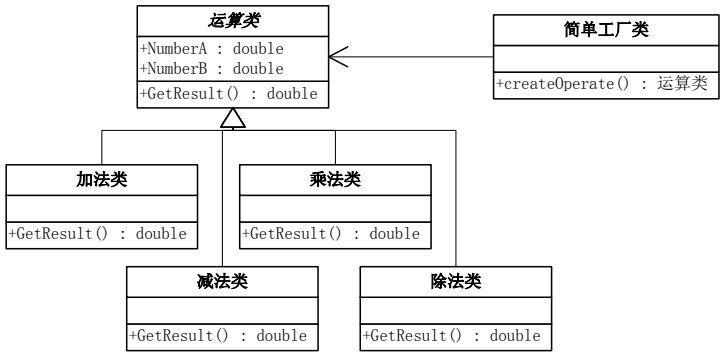
大鸟：“嗯？够了吗？”

小菜：“对了，还需要去修改运算类工厂，在 switch 中增加分支。”

大鸟：“哈，那才对，那如果要修改界面呢？”

小菜：“那就去改界面呀，关运算什么事呀。”

大鸟：“我们来看看这几个类的结构图。”

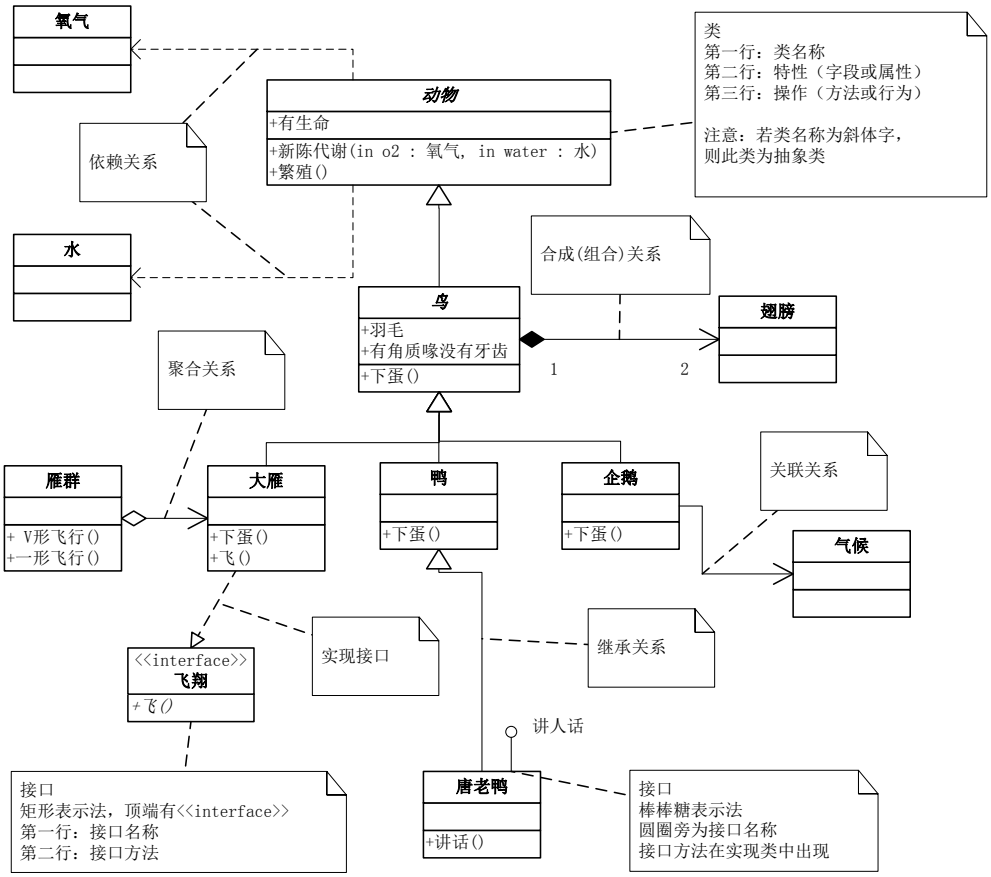


1.11 UML 类图

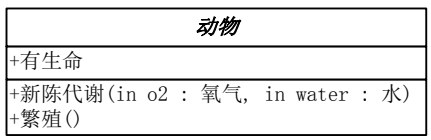
小菜：“对了，我时常在一些技术书中看到这些类图表示，简单的还看得懂，有些标记我很容易混淆。要不你给我讲讲吧。”

大鸟：“这个其实多看多用就熟悉了。我给你举一个例子，来看这样一幅图，其中就包括了 UML 类图中的基本图示法。”

UML 类图图示样例



大鸟：“首先你看那个‘动物’矩形框，它就代表一个类（Class）。类图分三层，第一层显示类的名称，如果是抽象类，则就用斜体显示。第二层是类的特性，通常就是字段和属性。第三层是类的操作，通常是方法或行为。注意前面的符号，‘+’表示 public，‘-’表示 private，‘#’表示 protected。”

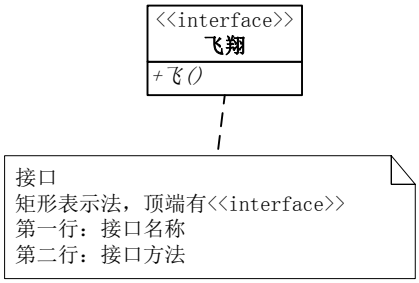


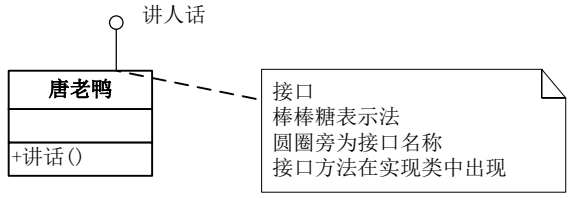
大鸟：“然后注意左下角的‘飞翔’，它表示一个接口图，与类图的区别主要是顶端有<<interface>>显示。第一行是接口名称，第二行是接口方法。接口还有另一种表示方法，俗称棒棒糖表示法，就是唐老鸭类实现了‘讲人话’的接口。”

小菜：“为什么要是‘讲人话’？”

大鸟：“鸭子本来也有语言，只不过只有唐老鸭是能讲人话的鸭子。”

小菜：“有道理。”



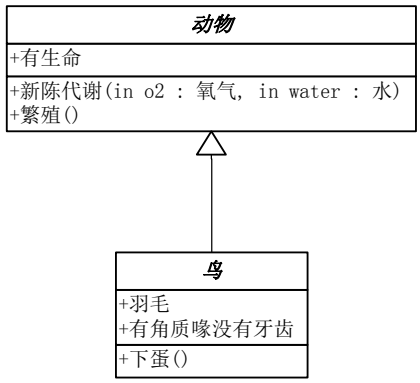



```
interface IFly
{
   void Fly();
}
```

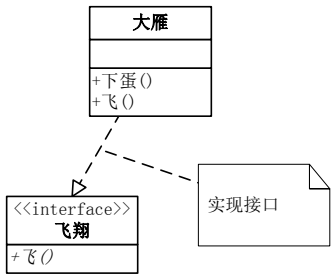
```
interface ILanguage
{
   void Speak ();
}
```

大鸟：“接下来就可讲类与类，类与接口之间的关系了。你可首先注意动物、鸟、鸭、唐老鸭之间关系符号。”

小菜：“明白了，它们都是继承的关系，继承关系用空心三角形+实线来表示。”



大鸟：“我举的几种鸟中，大雁是最能飞的，我让它实现了飞翔接口。实现接口用空心三角形+虚线来表示。”



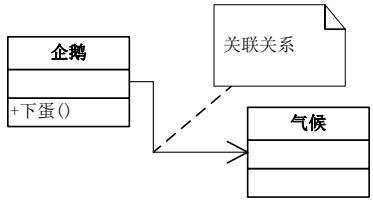

```
class Bird : Animal
{

}
```

```
class WideGoose : IFly
{

}
```

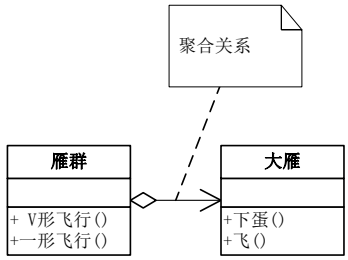
大鸟：“你看企鹅和气候两个类，企鹅是很特别的鸟，会游不会飞。更重要的是，它与气候有很大的关联。我们不去讨论为什么北极没有企鹅，为什么它们要每年长途跋涉。总之，企鹅需要‘知道’气候的变化，需要‘了解’气候规律。当一个类‘知道’另一个类时，可以用关联（association）。关联关系用实线箭头来表示。”




```
class Penguin : Bird
{
   private Climate climate;
}
```

在企鹅 Penguin 中，引用到气候 Climate 对象

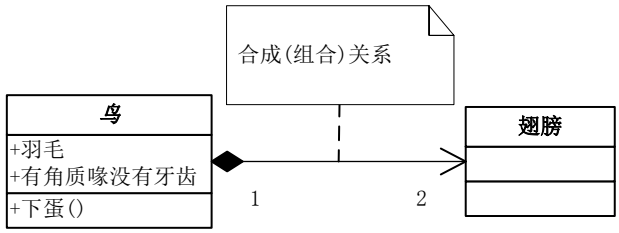
大鸟：“我们再来看大雁与雁群这两个类，大雁是群居动物，每只大雁都是属于一个雁群，一个雁群可以有多只大雁。所以它们之间就满足聚合（Aggregation）关系。**聚合表示一种弱的‘拥有’关系，体现的是 A 对象可以包含 B 对象，但 B 对象不是 A 对象的一部分**[DPE]（DPE 表示此句摘自《设计模式》（第 2 版），详细摘要说明见附录二）。聚合关系用空心的菱形+实线箭头来表示。”




```
class WideGooseAggregate
{
   private WideGoose[] arrayWideGoose;
}
```

在雁群 WideGooseAggregate 类中，有大雁数组对象 arrayWideGoose

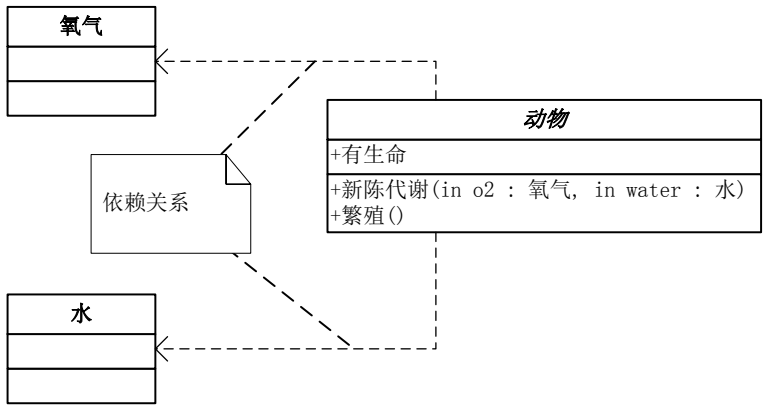
大鸟：“**合成（Composition，也有翻译成‘组合’的）是一种强的‘拥有’关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样**[DPE]。在这里鸟和其翅膀就是合成（组合）关系，因为它们是部分和整体的关系，并且翅膀和鸟的生命周期是相同的。合成关系用实心的菱形+实线箭头来表示。另外，你会注意到合成关系的连线两端还有一个数字‘1’和数字‘2’，这被称为基数。表明这一端的类可以有几个实例，很显然，一个鸟应该有两只翅膀。如果一个类可能有无数个实例，则就用‘n’来表示。关联关系、聚合关系也可以有基数的。”




```
class Bird
{
   private Wing wing;
   public Bird()
   {
        wing = new Wing();
   }
}
```

在鸟 Bird 类中，初始化时，实例化翅膀 Wing，它们之间同时生成

大鸟：“动物几大特征，比如有新陈代谢，能繁殖。而动物要有生命力，需要氧气、水以及食物等。也就是说，动物依赖于氧气和水。他们之间是依赖关系（Dependency），用虚线箭头来表示。”




```
abstract class Animal
{
   public Metabolism (Oxygen oxygen,Water water)
   {

   }
}
```

小菜：“啊，看来 UML 类图也不算难呀。回想那天我面试题写的代码，我终于明白我为什么写得不成功了，原来一个小小的计算器也可以写出这么精彩的代码，谢谢大鸟。”

大鸟：“吼吼，记住哦，**编程是一门技术，更加是一门艺术**，不能只满足于写完代码运行结果正确就完事，时常考虑如何让代码更加简练，更加容易维护，容易扩展和复用，只有这样才可以真正得到提

高。写出优雅的代码真的是一种很爽的事情。UML 类图也不是一学就会的，需要有一个慢慢熟练的过程。所谓学无止境，其实这才是理解面向对象的开始呢。”