

Head First C#中文版第一章

第一章 C#令你高效率：十分钟之内创建可视化应用程序



想要快速构建很棒的程序？

有了 C#，就相当于手头有了一门强大的编程语言和一个有价值的工具。有了 Visual Studio IDE，你让一个按钮工作再也不用花费好几个小时来写晦涩的代码了。更好的是，你可以把精力集中在你的工作上，而不用记住哪个方法参数是给按钮用的，哪个又是给标签（label）用的了。听起来很吸引人吧？翻到下一页，我们开始编程吧。

为什么要学 C#

C#和 Visual Studio IDE 帮你把写代码这事儿变得又简单又快捷。你用 C#工作的时候，VS IDE 就是你最好的朋友和长久的伴侣。

VS IDE-或者说是Visual Studio集成开发环境-是用C#工作的重要组成部分。它是一个帮你编辑代码、管理文件、发布项目的程序。

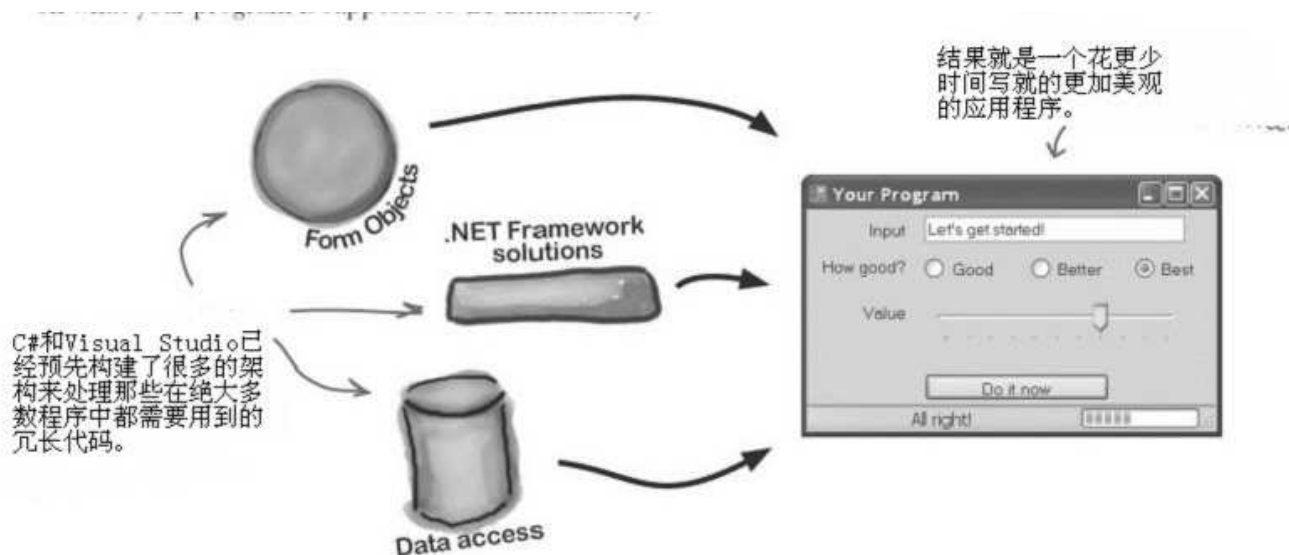
下面是 VS IDE 为你自动生成的...

每次你想要开始写一个程序，或者只是在窗体上放一个按钮，你的程序都需要一大堆的重复代码。



你能从 Visual Studio 和 C#这儿获得什么呢...

有了 C#这种专为 Windows 编程设计的语言和 Visual Studio 这种 IDE，你可以很快的专注于你期望你的程序要做的事情上。



C#和 Visual Studio IDE 让很多事情变得简单

使用 C#和 VS，你就得到了所有那些优秀的特性，无需做任何额外的工作。

这些特性使得你可以：

快速构建应用。用 C#创建程序是小菜一碟。C#好学而且强大，VS 又可以自动为你完成很多工作。你可以把俗气的代码交给 VS 去完成，自己把精力集中在要完成的事上。

设计美观的 UI。VS 的窗体设计器是最易用的设计工具。它为你完成许多事情，你简直就会

觉得创建很炫的 UI 是开发 C#应用中最令人满足的事儿。你可以创建功能完善的、专业的程序，而无需花费数小时从零开始的去写 GUI。

创建数据库并与之交互。VS 包含有创建数据库的建议接口，它与 SQL Sever Express 及其他一些流行的数据库系统无缝集成。

专注于解决你真正的问题。VS 确实为你做了很多工作，但你仍能掌控你用 C#创建的东西。

VS 让你专注于程序、工作（或叫做乐趣!）、客户。VS 负责单调乏味的苦差使，比如说：

管理所有的项目

使编辑项目的代码变得简单。

管理所有项目的图片、音频、图标及其它资源。

管理数据库并与之交互。

这意味着你可以把花在琐碎事务上的时间花在构建杀手级程序上。

你马上就会知道这话什么意思了。

帮助 CEO 实现无纸化

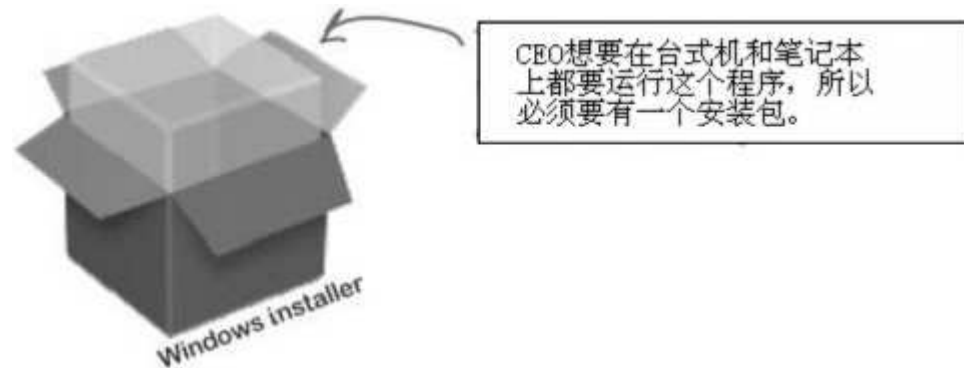
Objectville Paper 公司刚刚雇佣了一个新的 CEO。他喜欢徒步旅行、喝咖啡、热爱自然... 并且，他决定要尽力拯救森林。他想要从他的会晤开始做一个无纸化的主管。他周末要去白杨林去滑雪，他希望回来时可以看到新的通讯簿程序。要不然...那个老 CEO 也不用正在找新工作了。



在开始构建程序之前了解用户的需求

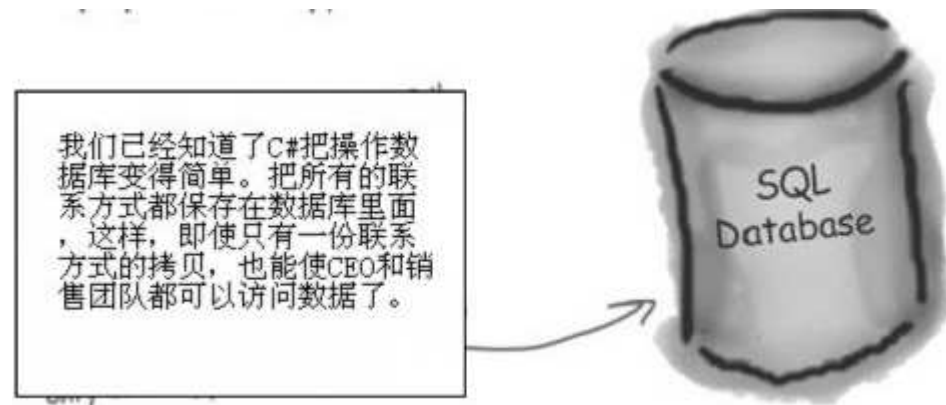
在开始写这个通讯簿的程序--或者是任何的程序--之前我们要花点时间去想想谁会去用这个程序，他们需要这个程序做什么？

1.CEO 需要在工作的时候用这个通讯簿程序，在他的笔记本电脑上也需要用。他需要一个安装包来确保把所有的文件都正确的安装到这两台电脑上。



2.Objectville Paper 公司的销售团队也想要用 CEO 的通讯簿。他们需要 CEO 的数据来建立一个邮件列表，来获得用户、获得更多的销售额。

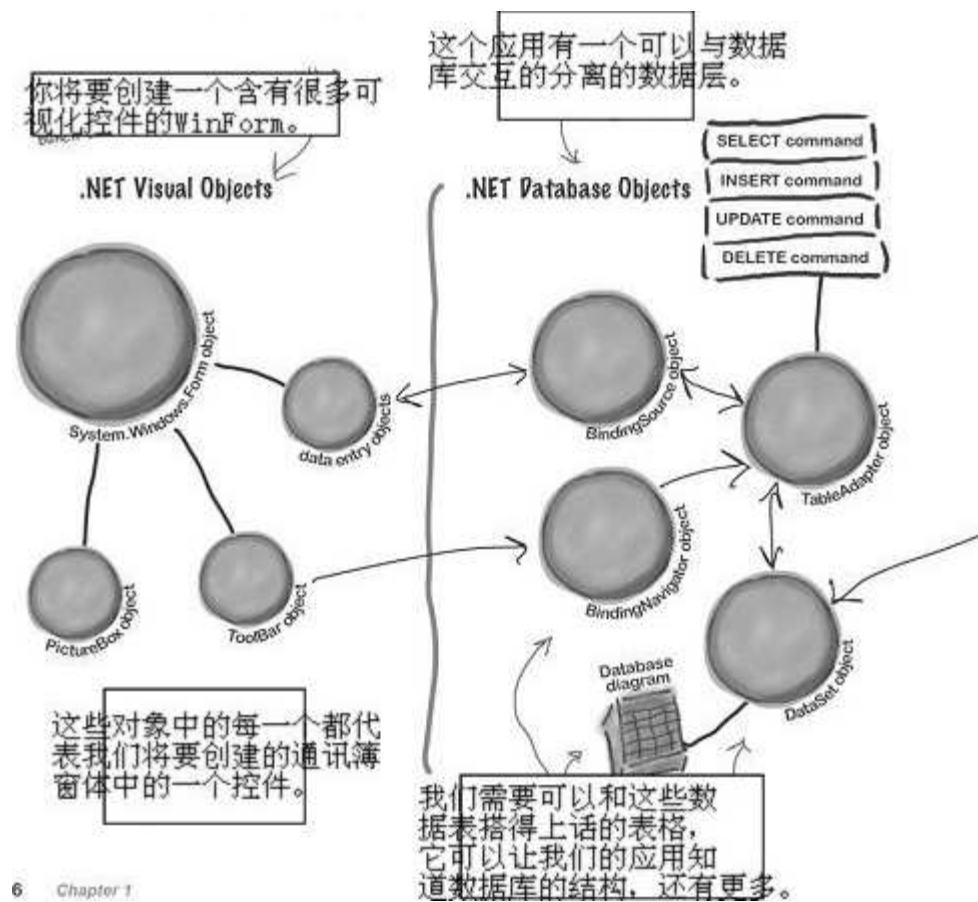
CEO 感觉到使用数据库是让全公司的人都可以看到他的数据的最佳途径，然后他就只需要维护所有联系方式的一份拷贝就行了。



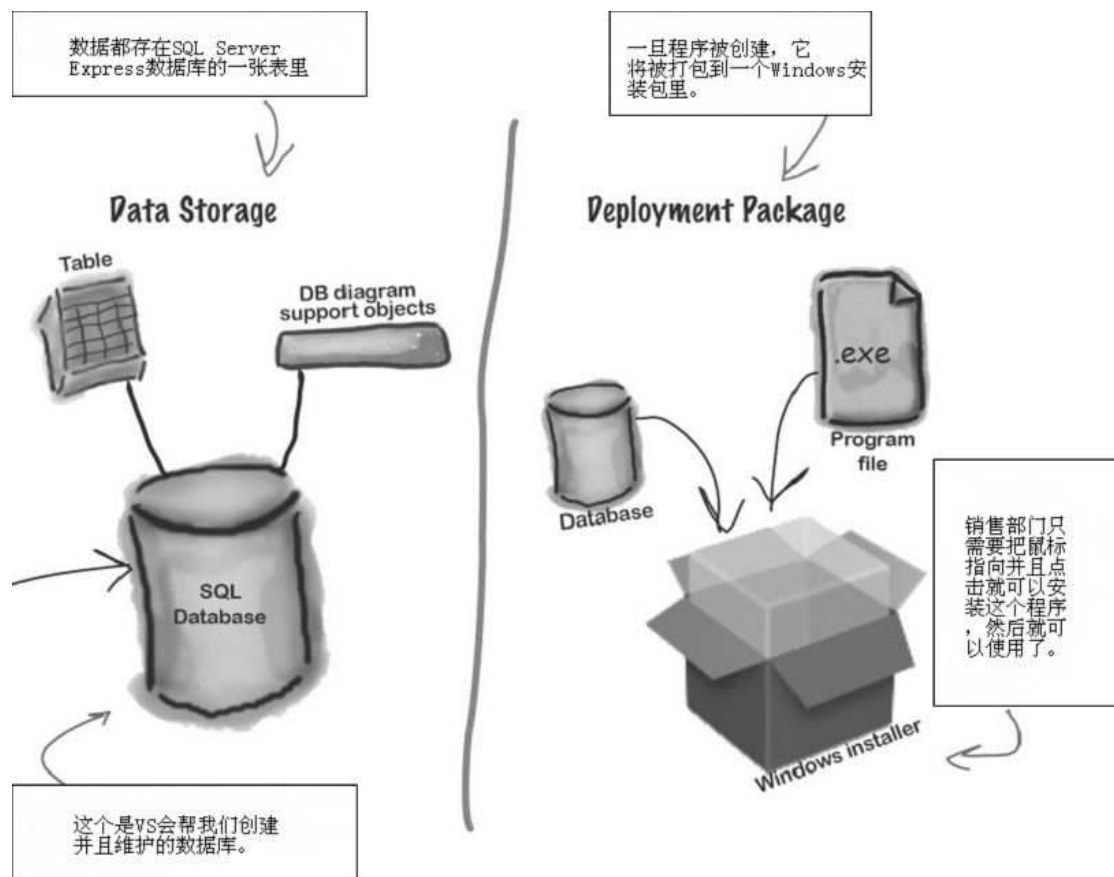
下面这些是你需要创建的

你需要一个有 GUI 的应用程序、一个可以和数据库说的上话的对象、数据库本身。还有一个安装包。听起来工作量很大，但是读完下面这几页你就可以创建完这些。

下面是我们要创建的程序的结构：

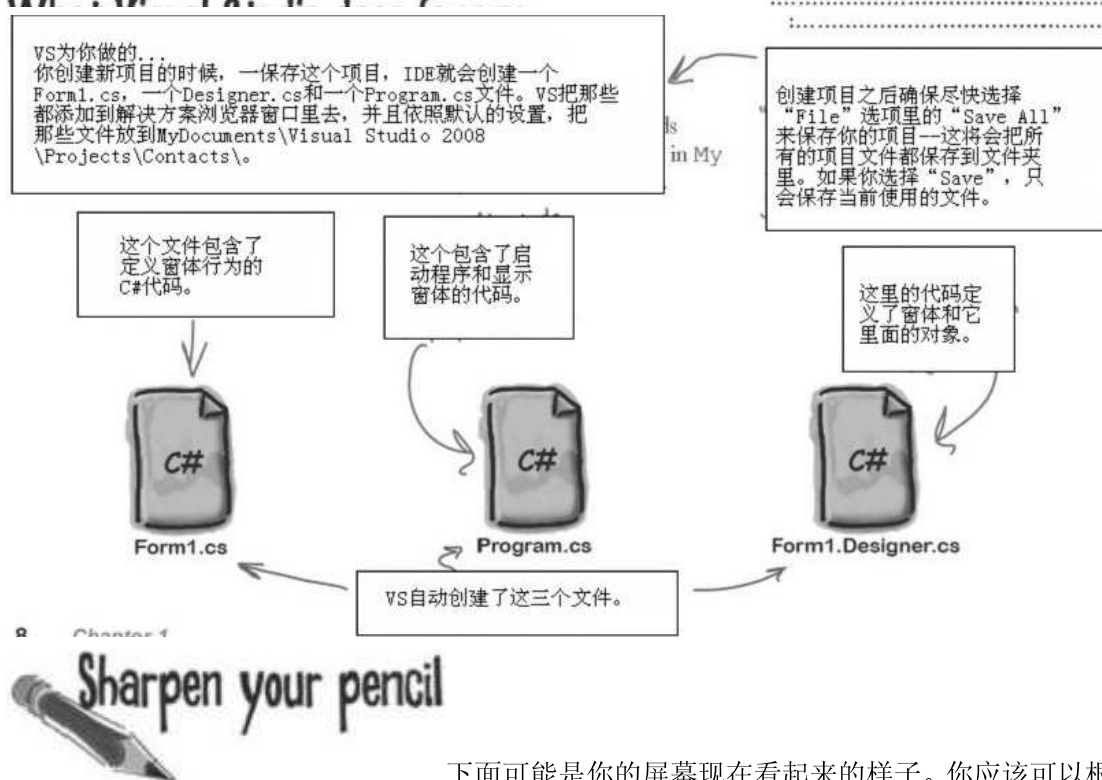
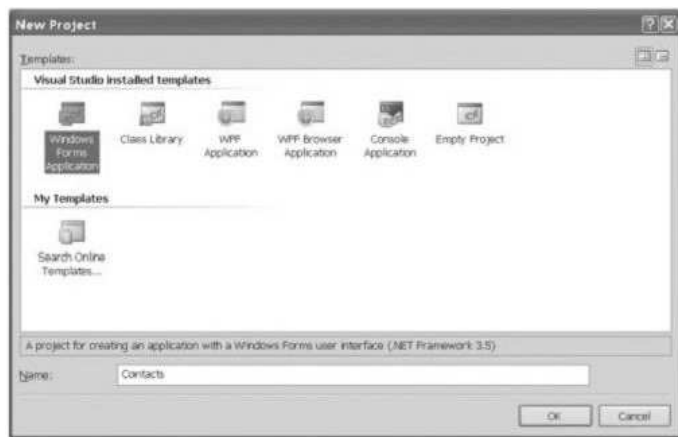


6 Chapter 1

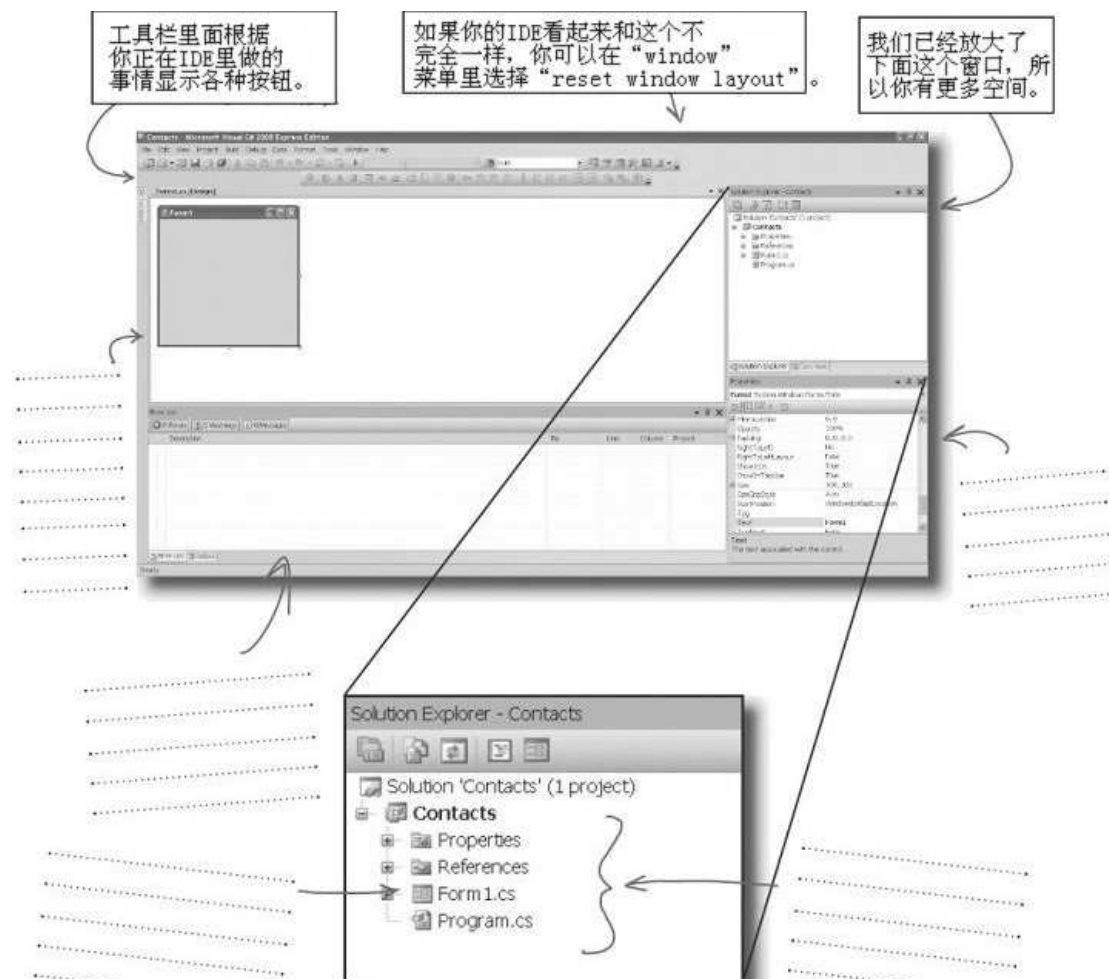


你需要在 VS 里面做的...

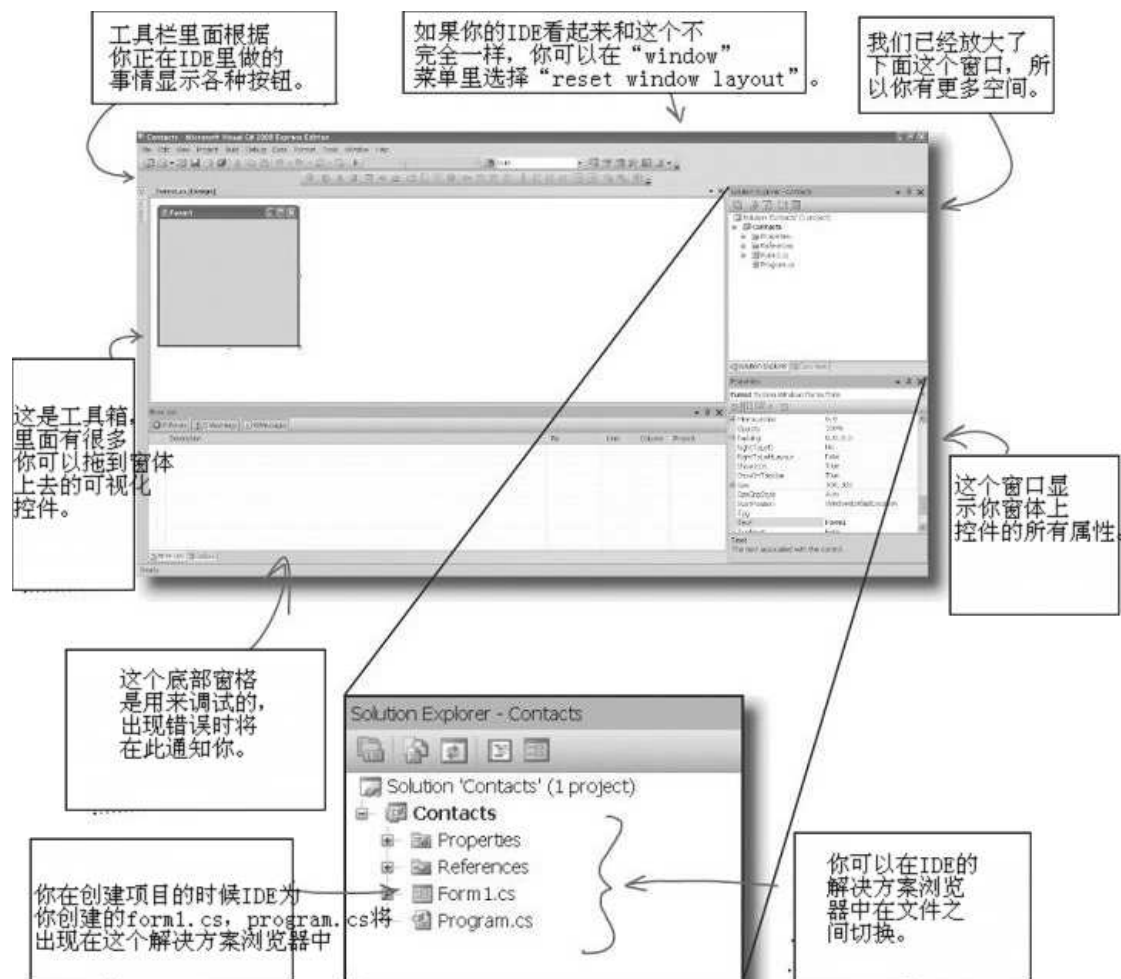
径直去打开 VS，如果你从没打开过 VS。跳过开始页在文件菜单里选择新项目。把你的项目命名为“Contacts”并点击确定。



下面可能是你的屏幕现在看起来的样子。你应该可以根据你已经知道的来分辨出来这些窗口和文件是什么。试着在每一个空白处填写关于这部分 IDE 是做什么的注释。我们已经做了一个来帮你开始。



我们已经填写了关于 VS IDE 不同部分的注释。你写的和我们或许不同，但是你应该已经可以基本分辨出每个窗口、每个部分是做什么用的。



问：如果 IDE 为我写了那么多代码的话，学 C#就是学怎么用 IDE 喽？

答：不，IDE 在自动生成代码方便很棒，但是它也只能做那么多了。有些事情它确实很在行，比如为你设置好的起点，自动改变窗体上控件的属性。但是编程中最难的部分--分辨出你的程序需要做什么而且让它去做--是任何 IDE 都做不到的。即使 VS 是最先进的 IDE 之一，它也只能做那么多了。写关键代码的是你而不是 IDE。

问：我在 VS 里面新建一个项目，但是进入“My Documents”下的“Projects”文件夹的时候，却没有发现它。怎么回事？

答：首先，你肯定在用 VS2008--在 VS2005 中，不会发生的。当你用 VS2008 第一次创建一个项目的时候，IDE 在 Local Settings\Application Data\Temporary Projects 文件夹下创建了项目文件。当你第一次保存项目时，IDE 提示你输入新文件名，并把它保存在 My Documents\Visual Studio 2008\Projects 文件夹下。当你试着打开一个新项目或者关闭一个临时的时，你将会被提示去保存或者放弃这个临时项目。

问：如果 IDE 创建了我不要的代码怎么办？

答：你可以改变它。IDE 被设置为根据被拖进或添加的元素的通常用法来创建代码。但是有时那不是你想要的。IDE 为你做的每一件事--它创建的每行代码，它添加的每个文件--是可以修改的，或者手动直接修改文件或者通过一个使用方便的 IDE 的接口。

问：我下载 VS 的 Express 版本可以吗？或者我一定要用付费版本的才可以跟着这本书做？

答：这本书上没有免费版（从微软网站下载的版本）不能做的事情。Express 版本和其他版本（专业版、团队版）的不同是不会阻碍写 C# 代码和创建完整的功能，完善的应用。

问：可以重命名 IDE 创建的文件的文件名吗？

答：当然，你可以改变你程序的任何一方面。但是 IDE 是智能感应的命名文件的。当你添加一个文件时，你选择的文件名会影响代码生成的方式，创建的代码里会包含这个文件名。一些情况下，如果你重命名文件，你就得去修改代码，或者接受代码中和文件名的不同。那有点儿不爽，建议你除非必要不要修改文件名。

问：我的 IDE 看起来和你们的不一样！有些窗口没有，有些位置不对，咋回事啊？

答：如果你点击“窗口”菜单下的“重置窗口布局”，IDE 将会为你恢复窗口默认布局。然后你的就看起来和我们的一样了。

VS 会生成让你当做你的应用的起点的代码。

记住，程序实现预期的表现还是取决于你的。

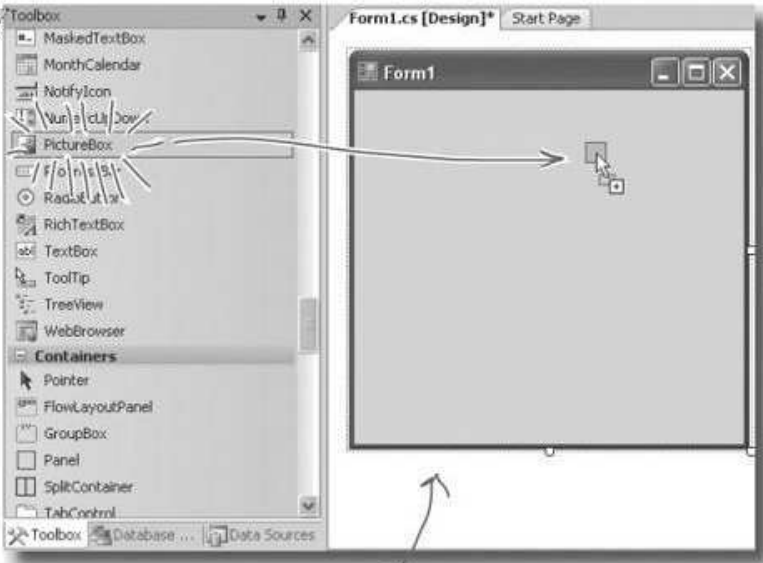
一图抵千语

开发 UI

有了 VS IDE，添加控件和润饰 UI 就简单到只需要拖拽的程度。我们来给窗体上添加一个 logo：

1 用PictureBox控件添加一张图片。在工具箱里面点击PictureBox控件并把它拖到窗体上去。VS IDE会在后台向Form1.Designer.cs中添加代码创建PictureBox控件。

如果你看不到工具箱，试着在IDE左上方的“工具箱”几个字上方悬停鼠标。如果没有那几个字，在“视图”菜单里点击“工具箱”使之出现。

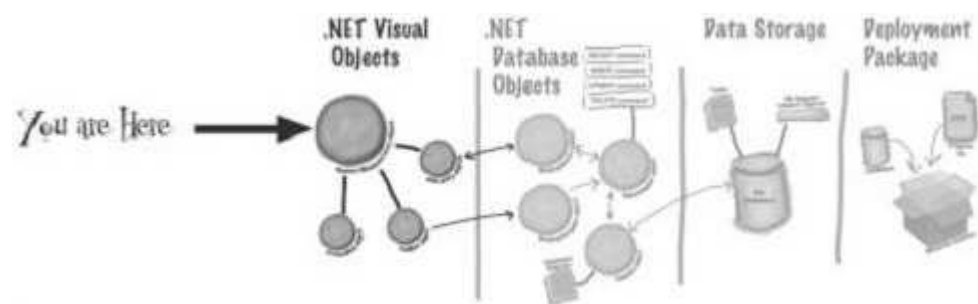


每次你修改一个控件的属性，IDE都会相应修改Form1.Designer.cs里面的代码。

如果你对UI设计并不在行也没关系。我们后面还要学很多关于设计好的UI事情。

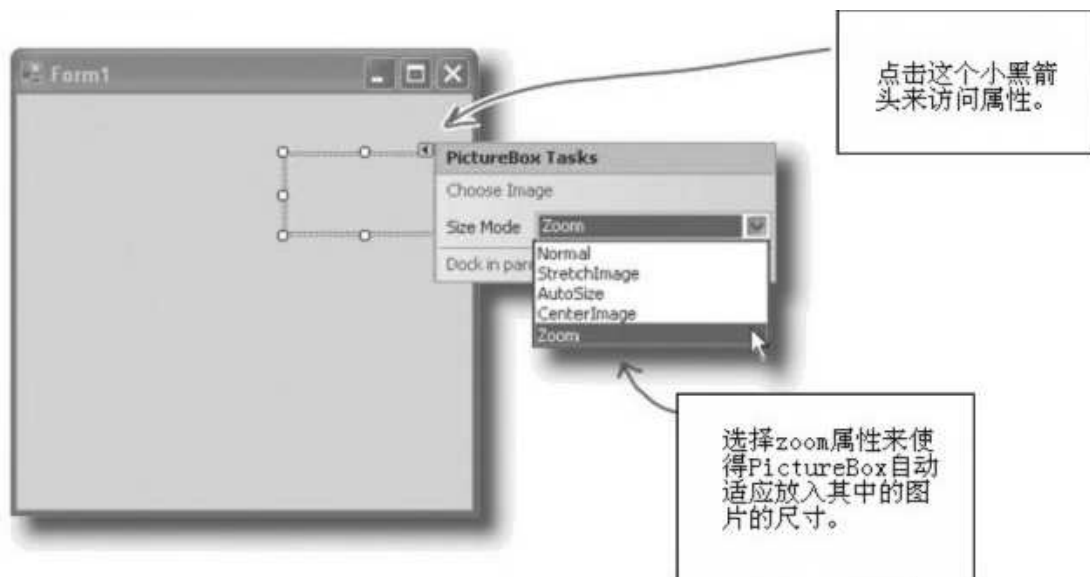
现在，把图片logo和其他控件管好，关心程序的行为。接下来我们要添加一些样式。

Relax



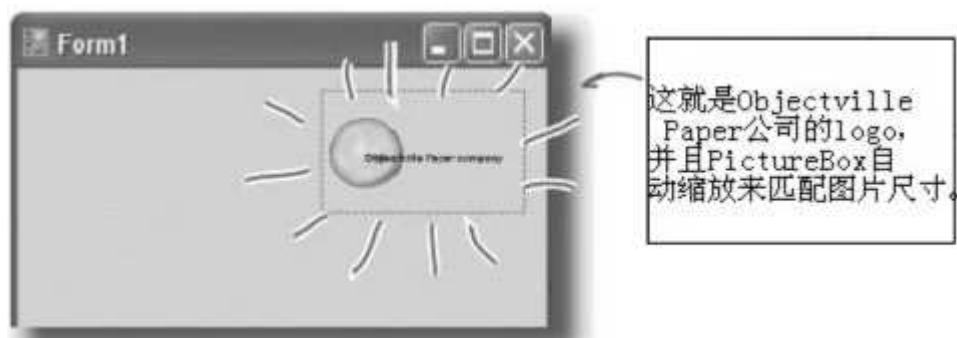
2 把 PictureBox 设置为 zoom 模式。

你窗体上的每一个控件都有你可以设置的属性。点击下图中的小黑剑头儿就可以访问那些属性。把 PictureBox 的 Size 属性设置为 zoom 看看会怎样。



3 下载 Objectville Paper 公司的 logo。

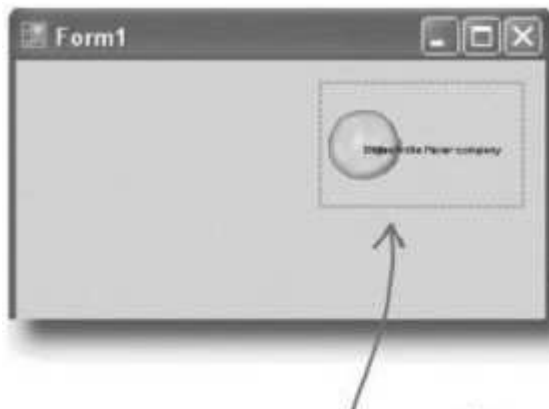
从 Head First 实验室 下载 Objectville Paper 公司的 logo (<http://www.headfirstlabs.com/books/hfcsharp>) 并保存到你的硬盘。然后点击 PictureBox 的属性箭头，并选择图片。点击 Import，找到你的 logo，你就完工了：



屏幕后的 Visual Studio

每次你用 VS IDE 做什么，IDE 都会为你写代码。当你创建了这个 logo 并告诉 VS 使用你下载的图片，VS 会创建一个资源并把它与你的应用关联起来。资源指的是任何的图像文件、音频文件、图标、或者任何其他与你的应用的捆绑的任何数据文件。图像文件与程序集成，所以当程序安装到其他的机器时，图像会一起被安装而且 PictureBox 可以使用它。

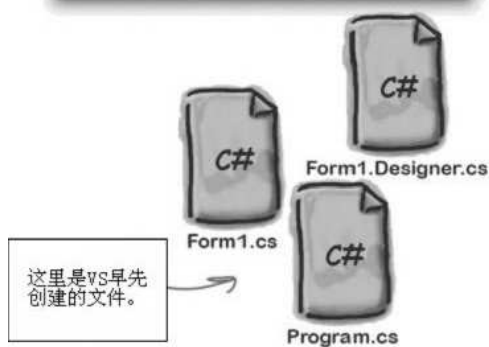
当你把 PictureBox 拖拽到你的窗体，IDE 会自动的创建一个叫做 Form1.resx 来存储资源并把它保存在项目里。双击这个文件，你可以看见新近添加的图片。



现在这个图片是通讯簿程序的一个资源。



如果你在解决方案浏览器里点击 Form1.resx, 你可以看到你包含的那个图片logo。Form1.resx连接PictureBox和图片, VS添加代码来实现连接。



这里是VS早先创建的文件。



当你包含一张图片, IDE创建Form1.resx文件。它包含所有与应用有关的资源(图片、音频, 和其他种类的数据文件)。

增加自动生成的代码

IDE 为你创建了很多代码, 但是你仍然会想要深入代码并增加它。让我们设置 logo, 让它在被用户双击的时候显示关于信息。

```

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void pictureBox1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Contact List 1.0.\nWritten by: Your Name", "About");
    }
}

```

当你双击这个PictureBox控件的时候，VS为你创建这个箭头指向的方法。每次用户在程序运行时点击这个PictureBox控件方法就会运行。

这个方法名向你表明了它什么时候运行：当有人点击这个PictureBox的时候。

当你双击PictureBox的时候VS会打开代码，并把闪烁的光标定位到此处。不要管那些在你输入代码时IDE弹出的各种窗口，那些是用来帮助你的，但我们现在还用不上。

输入这行代码。它使得一个带有你提供的文本的消息框弹出。消息框会被标题为“About”。

一旦你输入了这行代码，通过点击IDE工具栏上的保存按钮或者选择文件菜单里的保存选项来保存代码。养成有规律的全部保存的习惯。

there are no
Dumb Questions

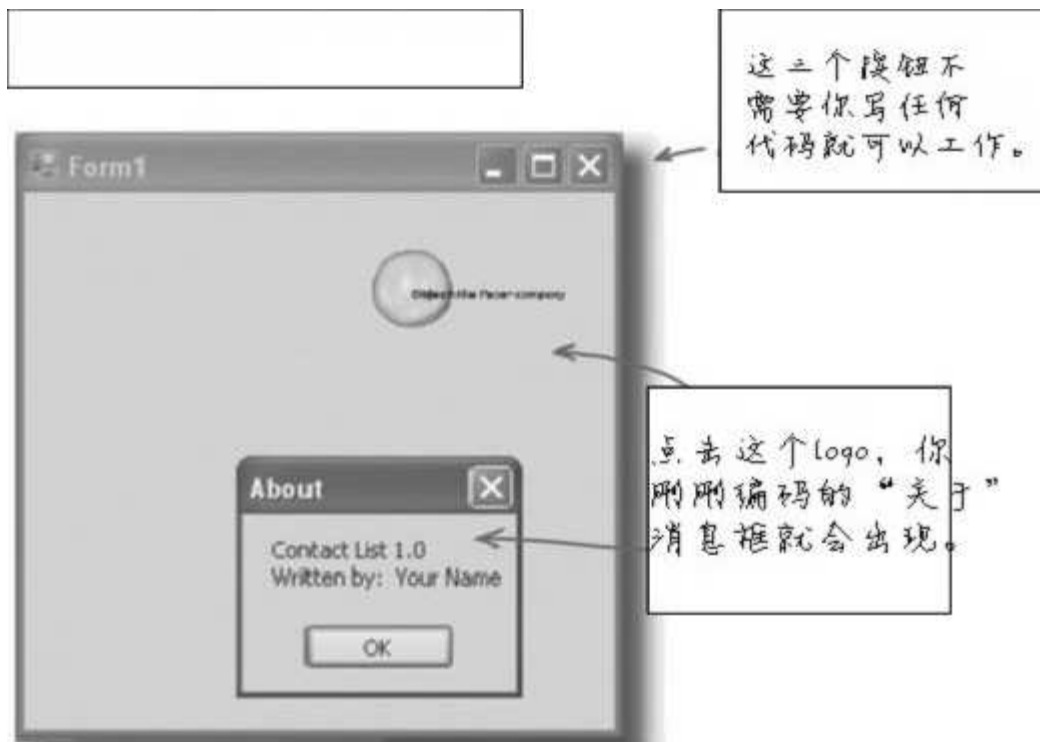
问：什么叫方法？

答：方法就是一个有名字的代码块。我们在第二章会讨论更多关于方法的事儿。

问：\n 是干什么用的？

答：那是换行符。它告诉 C# 把 “Contact List 1.0” 放在第一行，并为 “Written by” 新起一行。你**已经**可以运行你的应用了

按你键盘上的 F5 键，或者点击工具栏上的绿色按钮来检查目前为止你已经做了什么。（这被叫做 “调试”，意味着用 IDE 运行你的程序。）你可以通过选择 “调试” 菜单里的 “停止调试” 或者点击工具栏的按钮来停止调试。



我的文件在哪儿?

当你运行程序的时候VS会把你程序复制到My Documents\Visual Studio\Projects\Contacts\Contacts\bin\debug。你甚至可以直接转到文件夹, 双击exe文件来运行程序。

C#把你的程序放进一个你可以运行的文件, 叫做可执行文件。你可以在这里的debug文件夹下找到它。



这不是个错误, 有两层文件夹。内层放着真正的C#代码文件。

there are no
Dumb Questions

问: 在我的 IDE 里, 绿色的肩头被标为“调试”。这是个问题吗?

答: 不。调试, 至少对于我们的目的来说, 意味着在 IDE 里运行你的程序。我们后面会讨论很多关于调试的事儿, 但是现在, 你可以简单的认为它就是运行你的程序的方法。

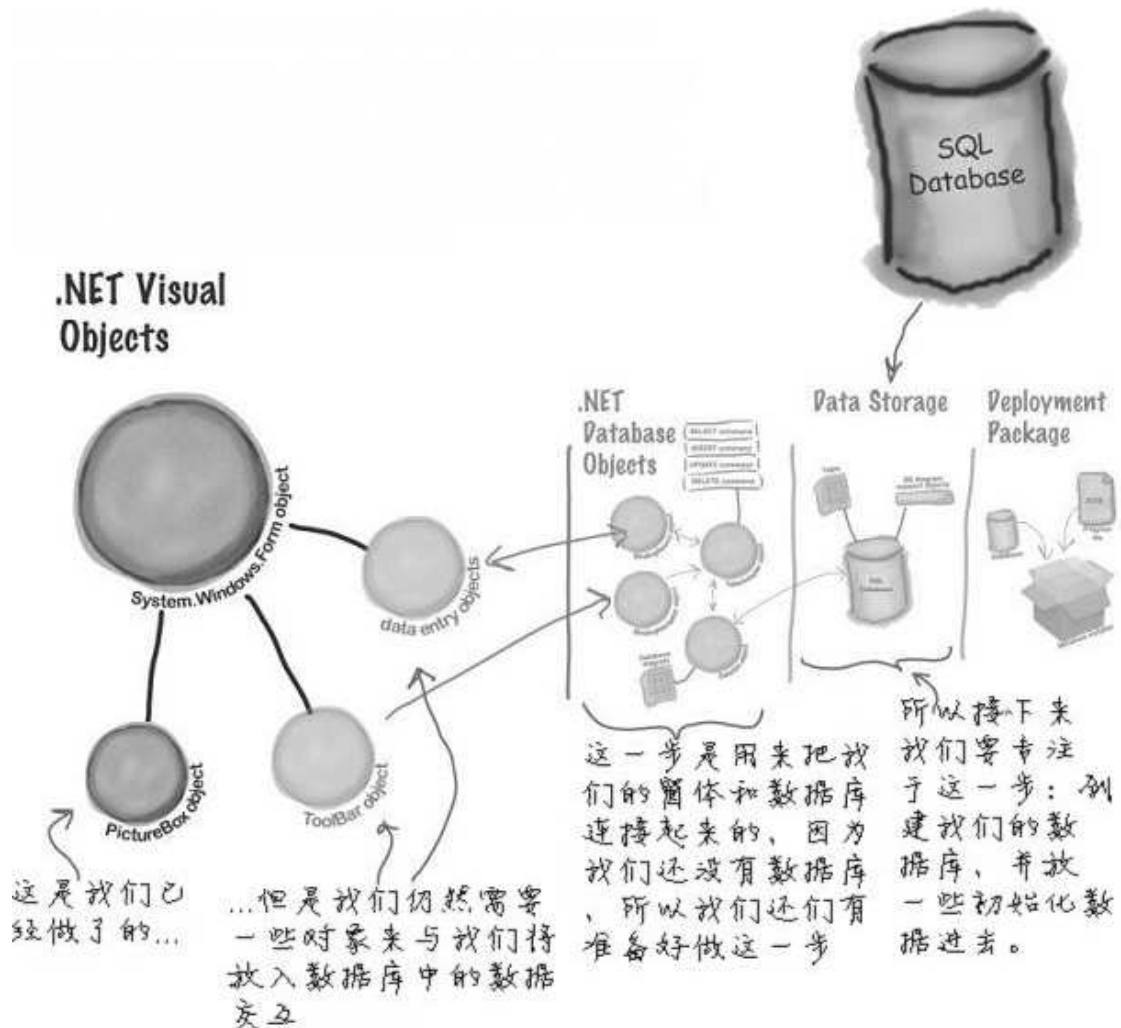
问: 在我的工具栏上我没看见停止调试按钮。怎么搞的?

答: 停止运行按钮只在你运行程序时才会出现在一个特殊的工具栏上出现。试着再运行一次程序, 看看它是不是会出现。

我们已经做了这么多了

我们已经建了一个窗体还有一个在被点击时会弹出消息框的 PictureBox 对象。接下来, 我们需要添加卡片上的其他字段, 比如联系人的名字和电话。

我们把那些数据存在一个数据库里。VS 可以为我们直接连接数据库的字段, 这意味着我们不用和很多的数据库访问代码打交道了 (那还挺好的)。要想工作, 我们要创建数据库, 这样窗体上的控件才能链接到它。所以我们就要从 .NET 可视化对象直接跳到数据存储的部分了。



VS 可以生成代码来把你的窗体和数据库连接起来, 但是你需要在生成代码之前先有数据库。

我们需要一个数据库来存储我们的信息

在向窗体添加剩下的字段之前, 我们需要创建一个数据库来和窗体连接。IDE 会创建很多代码来把数据和我们的窗体连接起来, 但是我们还是要先创建数据库本身。

确定在继续之前你停止了调试。

1 向你的项目添加一个 SQL 数据库

在解决方案浏览器里，右键点击 Contacts 项目，选择添加，然后选择新建项。选择 SQL 数据库图标，并把它命名为 ContactDB.mdf。



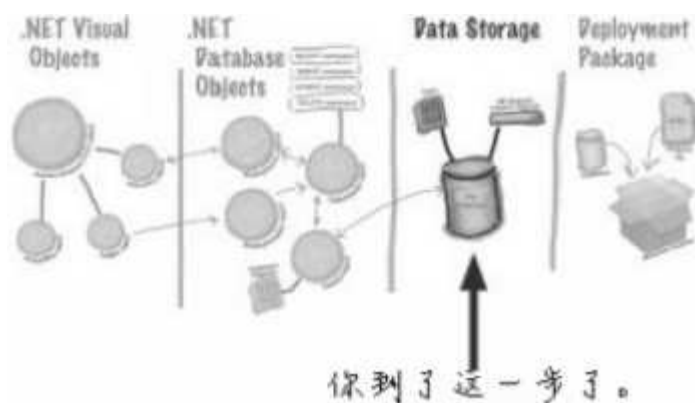
2 取消数据源配置向导。

现在，我们想要跳过配置数据源的步骤，所以，点击取消按钮。我们一设置完数据库结构就回到这一步来。

3 在解决方案浏览器里查看你的数据库。

在解决方案浏览器里你会看到 ContactDB 已经被添加到文件列表里。双击 ContactDB.mdf，看看你屏幕的左侧。工具箱已经变成了数据库浏览器。

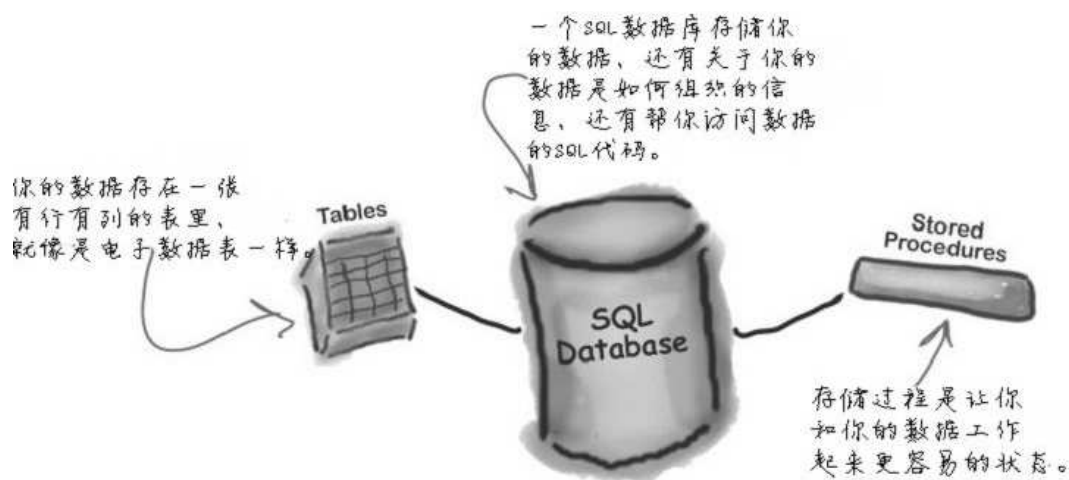




IDE 创建了一个数据库

你告诉 IDE 向你的项目添加一个新的数据库的时候，IDE 就为你创建了一个数据库。SQL 数据库是一个有组织的、有关联的方式为你存储数据的系统。IDE 给了你维护你的数据所需要的所有工具。

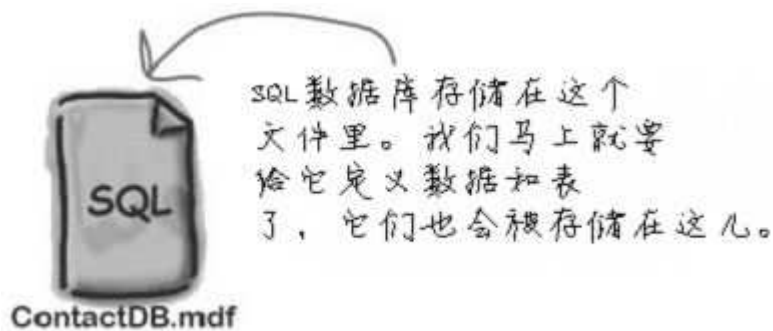
SQL 数据库里的数据存在表里。现在来说，你可以认为它是一个电子制表软件。它把你的数据组织在行列之中。列代表数据分类，比如一个联系人的名字和电话，每一行就代表一张联系人卡片上的数据。



SQL 是它自己的语言

SQL 代表结构化查询语言。它是一种用来访问数据库中数据的编程语言。它有自己的语法、关键字、和结构。SQL 代码以状态和查询的形式来访问和检索数据。一个 SQL 数据库可以保存代表很多 SQL 状态和查询的存储过程，存储过程保存在数据库里，并且可以随时运行。IDE 为你生成存储过程来让你的程序可以访问数据库里面的数据。

营销建议：我们可以在这儿加一个“Head First SQL”的链接吗？

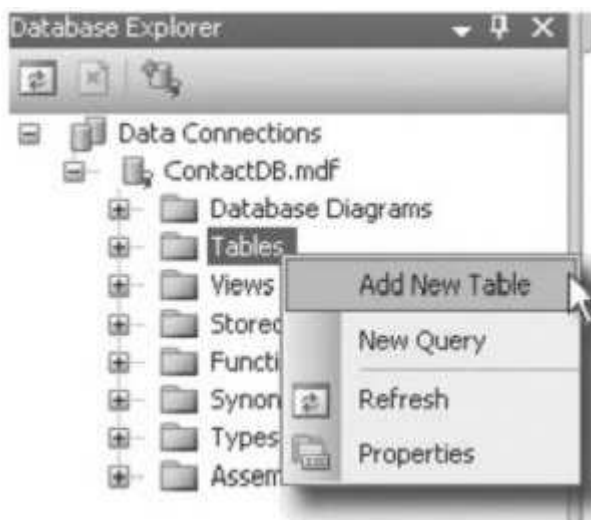


给联系人列表创建数据表

我们有一个数据库，现在我们需要往里面存储信息了。但是，实际上，我们的信息需要放进一个数据表里，数据表是数据库用来存储每一个 bit 数据的数据结构。对于我们的应用来说，我们来创建一个叫做“People”的数据表来存储所有的联系人信息。

1 向 ContactDB 数据库里面添加一个数据表。

在数据库浏览器里面右键单击 Tables，并且选择 Add New Table。这会打开一个让你来定义刚刚创建的数据表中数据列的窗口。



现在我们需要向我们的数据表中添加数据列了。首先，让我们向新建的 People 数据表中添加一个叫做 ContactID 的数据列，这样每一个联系人记录就会有一个唯一的 ID 了。

2 向 People 数据表中添加 ContactID 数据列

在列名字段中键入“ContactID”，并在数据类型下拉列表中选择 Int。一定要反选允许为空复选框。

最后，我们把它作为我们数据表的主键。选中你刚刚创建的 ContactID 数据列，并点击主键按钮。这告诉数据库每一个条目都将会有一个主键条目。



这个是主键按钮。主键帮助你的数据库更快的查找记录。

there are no Dumb Questions

问：什么是数据列来着？

答：数据列是数据库里面的一个字段。所以在 People 数据库里面，你可能会有 FirstName 和 LastName 数据列。它总会有一个数据类型，比如 String 或者 Date 或者 Bool。

问：为什么我们需要 ContactID 数据列呢？

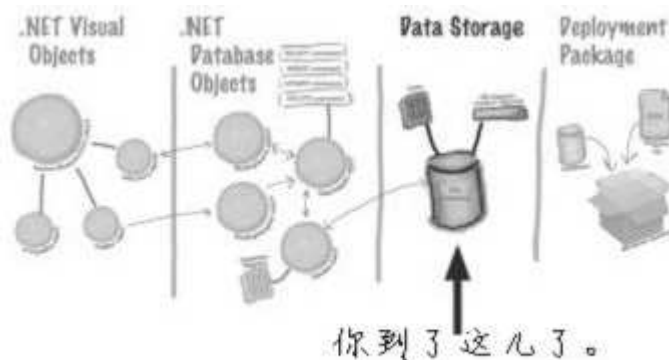
答：对于绝大多数的数据库来说有一个唯一的 ID 会大有裨益的。因为我们要为每一个人存储联系信息，所以我们决定给它创建一个数据列，并且称之为 ContactID。

问：数据类型里的那个 int 什么意思？

答：数据类型告诉数据库每一个数据列期待什么类型的信息。Int 代表 integer，它是一个整数。所以 ContactID 数据列将会存储整数。

问：东西太多了。我应该要都领会吗？

答：不，你现在还不能全明白也 OK。集中精力于基本步骤上，在这本书的后续章节里面我们会花更多的时间在数据库上。如果你现在非常渴望懂得更多，你总是可以拿起 Head First SQL 来和这本书一起读的。

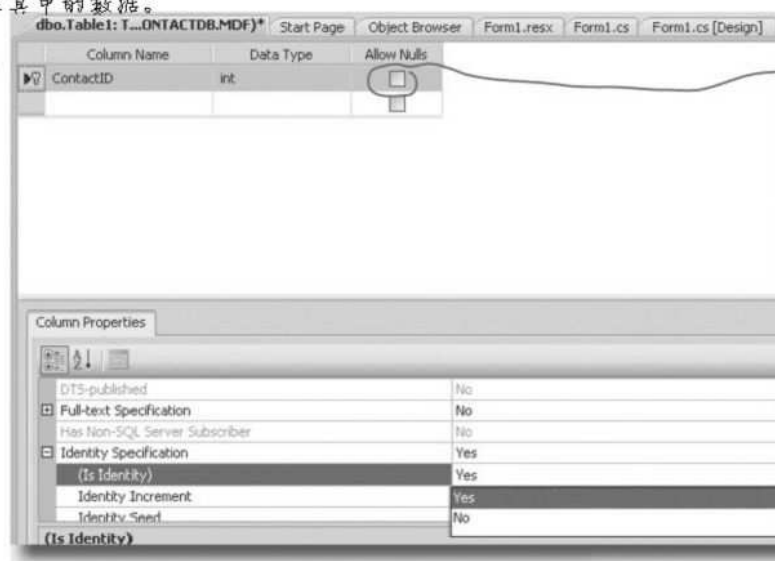


3 告诉数据库去自动生成 ID.

鉴于 ContactID 是数据库的一个数字，所以应该是我们而不是用户们来告诉我们的数据库去自动为我们的处理创建和分配 ID。这样，我们就无需写任何代码来实现了。

在你的数据表下面的属性栏里，向下滚动到 Identity Specification，点击按钮+，然后点击 (Is Identity) 属性旁的 Yes。

你用这个窗口来创建数据表并存储在其中的数据。



不要选中它是很重要的。因为主键是你的程序定位记录的主要方式，它总是需要有值的。

这将会使得 ContactID 字段一旦有新的记录被添加就会自动被更新。

联系人卡片上的空白就是我们的 People 数据表上的数据列

现在你已经给数据表创建了一个主键，你需要定义所有你要在数据库中跟踪的字段。我们写好的联系人卡片上的每一个字段应该对应 People 数据库里的一个数据列。



对于每一个人来说，我们想要存储他的名字、电话、Email地址，如果他是 Objectville Paper Company 公司的客户，还需要存储上次给他打电话的时间。

联系人卡片上的每一个字段应该对应 People 数据库里的一个数据列。



用多行数据为同一个人存储信息会产生什么后果？

WHO DOES WHAT?

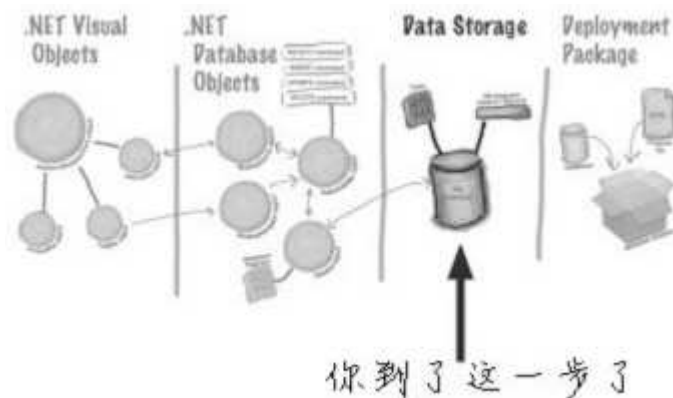
现在你已经创建了一个People数据表和一个主键列，你需要为所有的数据字段添加数据列了。看看你能不能分辨出哪种数据类型对应你的数据表中哪个数据列，并且和右侧的描述连线起来。

列名	数据类型	描述
Last Call	int	这种类型存储日期和时间
Name	bit	布尔类型
ContactID	nvarchar(50)	最长50的一串字母、数字和其他字符。
Client?	datetime	整数

WHO DOES WHAT?

现在你已经创建了一个People数据表和一个主键列，你需要为所有的数据字段添加数据列了。看看你能不能分辨出哪种数据类型对应你的数据表中哪个数据列，并且和右侧的描述连线起来。

列名	数据类型	描述
Last Call	int	这种类型存储日期和时间
Name	bit	布尔类型
ContactID	nvarchar(50)	最长50的一串字母、数字和其他字符。
Client?	datetime	整数



创建完数据表

把目光从联系人卡片移回到你创建 ContactID 和其他五个数据列的地方。下面是你的数据表完成时应该看起来的样子：

Column Name	Data Type	Allow Nulls
ContactID	int	<input type="checkbox"/>
Name	nvarchar(50)	<input checked="" type="checkbox"/>
Company	nvarchar(50)	<input checked="" type="checkbox"/>
Telephone	nvarchar(50)	<input checked="" type="checkbox"/>
Email	nvarchar(50)	<input checked="" type="checkbox"/>
Client	bit	<input checked="" type="checkbox"/>
LastCall	datetime	<input checked="" type="checkbox"/>

如果你反选 Allow Nulls，这一列就必须要有个值。

有的卡片可能会有一些缺失的信息，所以我们允许特定的列为空。

bit字段存储true或者false值，而且可以表现为一个复选框。

点击工具栏上的保存按钮来保存你的新数据表。你会被要求输入一个名字。称它为“People”并点击确认。

我们已经讨论过这个叫做“People”的数据表，但是直到这一步你才给它一个正式的名字。

这会创建一个存储在ContactDB数据库里面的People数据表。

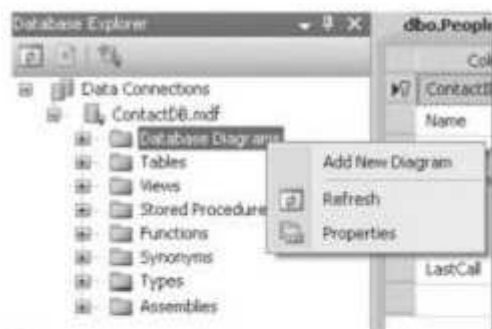
用图解法表示你的数据那样你的应用就可以访问它了

创建完了数据库和数据表，你需要让你的应用知道它。数据库图表就是从这儿来的。数据库图表是 VS IDE 可以用来和数据表协调工作的对于你的数据表的简单描述。它也使得 IDE 能

够自动生成 SQL 语句来添加、修改、和删除数据表里面的数据行。

1 创建新的数据库图表。

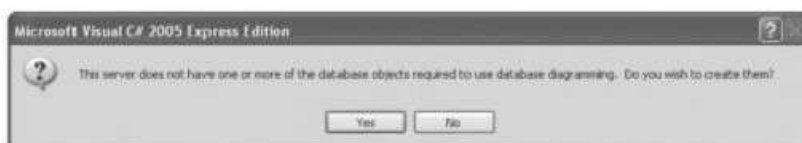
去数据库浏览器窗口并右键单击数据库图表节点。选择添加新图表。



记住，↑这些选项
都在ContactDB下，
所以他们都将应用于
那个特定的数据库。

2 让 IDE 生成访问代码。

在你告诉 IDE 你的特定数据表之前，它需要创建一些基本存储过程来与你的数据库互动。在这儿点击 Yes，并让 IDE 去工作。



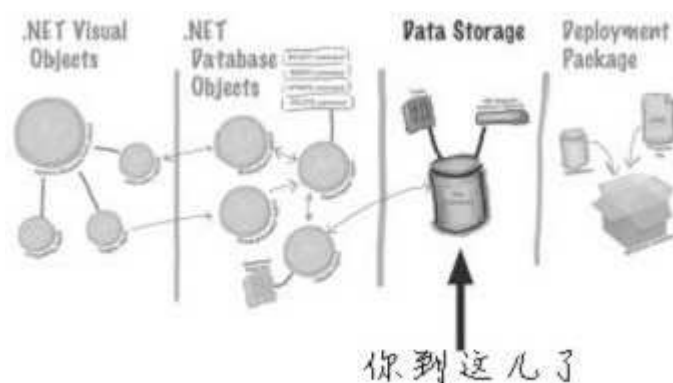
← IDE 创建一些
允许你的程序
与你创建的数
据库交互的存
储过程。

3 选择你要用的数据表。

在弹出的窗口中选择 People 数据表，并点击添加。现在 IDE 已经准备好了生成特定于你的数据表的代码。



← 当你有一个多数据
表的数据库时，每
一个数据表将会表
现为这个窗口上的
一个条目。



4 把你的图表命名为 PeopleDiagram。

选择文件>保存图表。你将被要求命名你的新数据库图表。称之为 PeopleDiagram，你就都搞定了。

← 如果你用 VS 2005，选择 文件>全部保存。

数据库图表在这儿被可视化的显示。它是对你的数据表的简化的表示。

如果你想要让数据库里的其他数据表被图示化，它们也将被显示在这儿。

dbo.Diagram1...NTACTDB.MDF)*		dbo.People: T...ON
People ContactID Name Company Telephone Email Client LastCall		

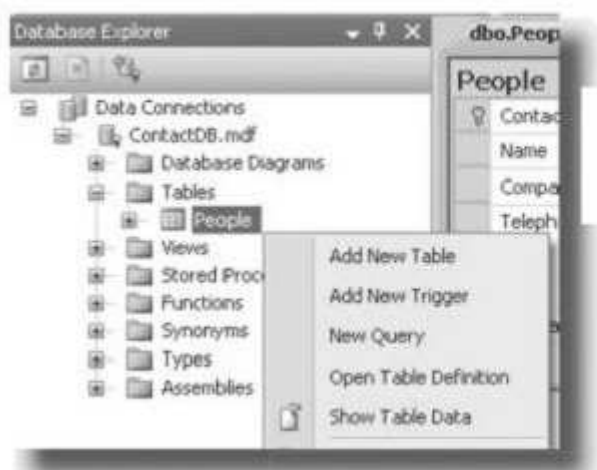
这只是你刚刚设计完成的数据库的一张截图它把 ContactID 标记为主键并且列出来数据库中所有其他的数据列。

数据库图表向你的 **VS IDE** 描述你的数据表。**IDE** 将会使用数据库图表来自动生成与数据库协调工作的代码。

向你的数据库插入你的卡片数据

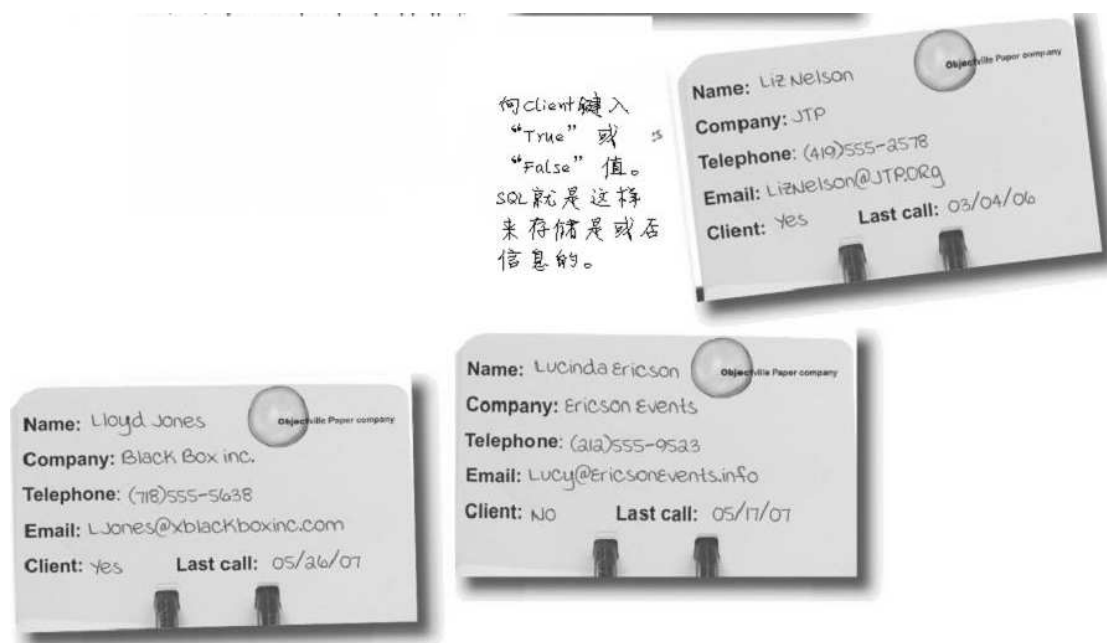
现在你已经准备好向数据库插入卡片。这里是老板的一些联系人信息--我们将用它们以几条数据来开始数据库。

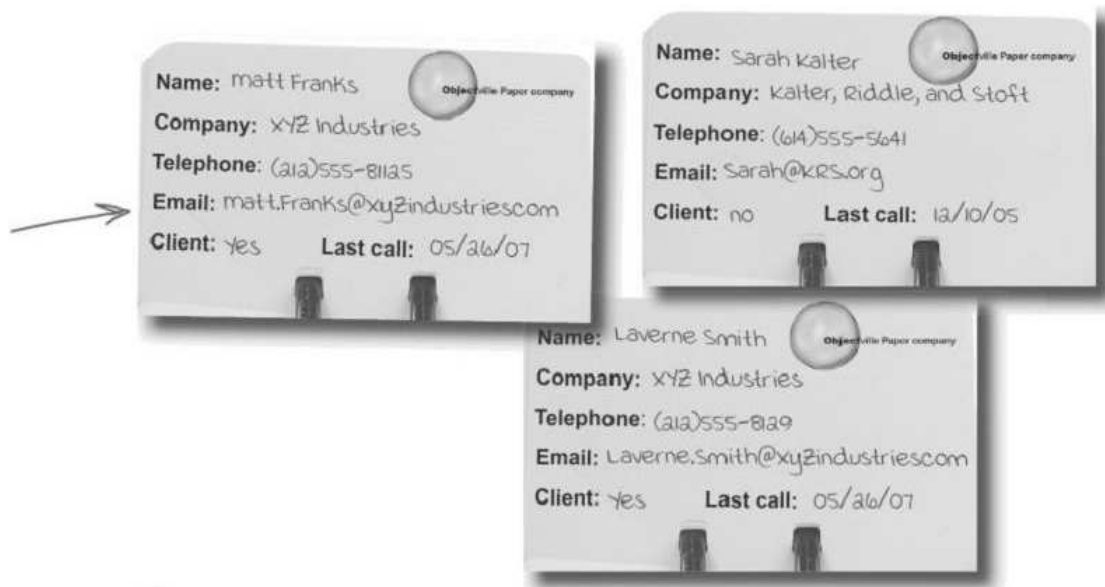
1 展开数据表然后在数据库浏览器（或者服务器浏览器）里面右键单击 **People** 数据表并选择显示数据表数据。



你的任务是向
People数据表
键入所有六张
卡片上的数据。

2 一旦你在主窗口中看见数据表格，径直去添加下面所有的数据吧。（一开始你将会看见所有的 NULL 值--当你添加你的第一行的时候覆盖它们就好了。并且忽略掉出现在数据旁边的感叹号。）无需你动手，ContactID 数据列将会自动填充。





3 一旦你键入了所有六条记录，再次从文件菜单选择全部保存。那将会把所有记录保存到数据库。

“全部保存”告诉IDE
去把你的应用中的每
一样都保存。那与只
保存当前文件的“保存”
是有区别的。

there are no
Dumb Questions

问：我输入完的数据怎么样了？它去哪儿了？

答：IDE 自动把你键入的数据存储进数据库的 People 数据表。数据表，它的列，数据类型，和里面的所有数据都存储在 SQL Server Express 的 ContactDB.mdf 文件里。这个文件被作为项目的一部分存储，并且像你的代码文件一样，IDE 会在你修改它的时候更新它。

问：好，我输入了列条记录。它们会永远作为我的程序的一部分吗？

答：对，它们就像你写的代码和正在创建的窗体一样，是你的程序的一部分。不同的是，他们不被编译为可执行文件，而是被复制并且和可执行文件一起存储。当你的程序需要访问数据时，它读写程序的输出目录下的 ContactDB.mdf。

这个文件实际是
一个SQL数据库，并
且你的程序可以
靠IDE为你生成的
代码来使用它。

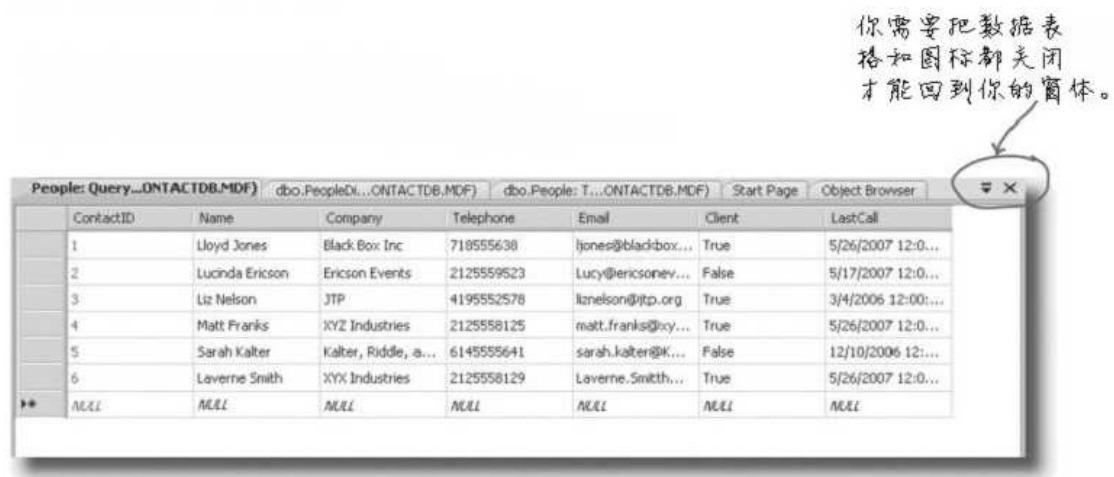


用数据源把你的窗体和你的数据库连接起来

我们终于准备好了让我们的窗体和数据库对话的.NET 数据库对象。我们需要数据源，那只是用来让你的程序和 ContactDB 数据库对话的一些 SQL 查询语句的集合。

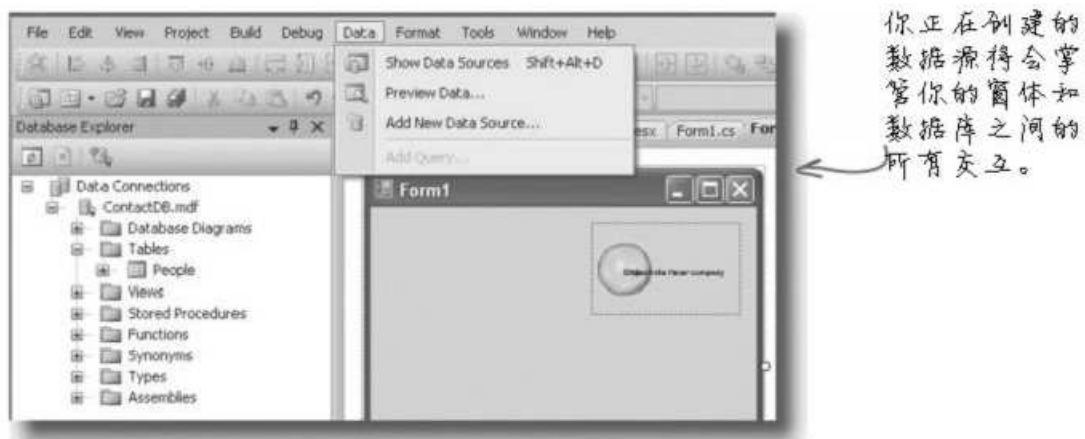
1 回到你的程序的窗体。

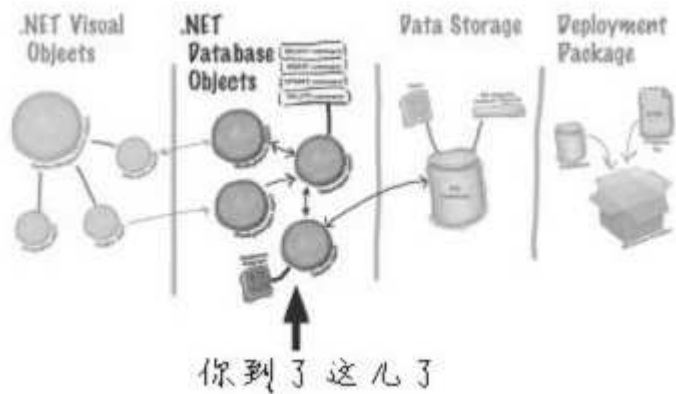
关闭 People 数据表和 ContactDB 数据库图表。你现在应该可以看见 Form1.cs[Design]了。



2 向你的应用添加一个数据源。

这个现在应该很简单了。点击数据菜单，然后从下拉列表选择添加新数据源。



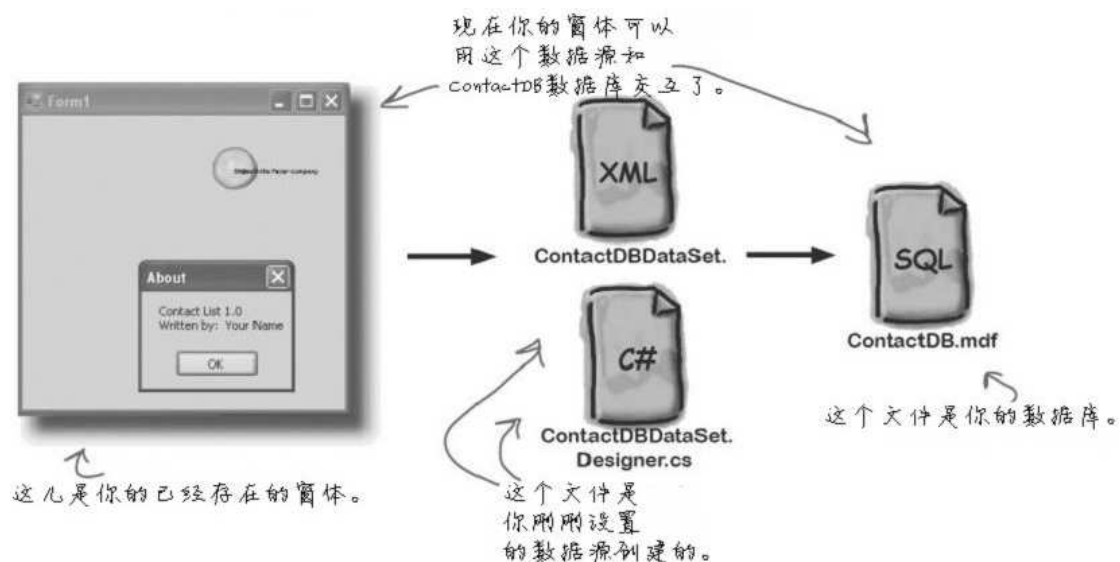


3 配置你的新数据源

现在你需要设置你的数据源来使用 ContactDB 数据库。下面是需要做的：

- *选择数据库并点击下一步按钮。
- *在“选择你的数据连接”这一屏点击下一步。
- *确定在接下来的“保存数据连接”这一屏勾选保存数据连接复选框并点击下一步。
- *在“选择你的对象”这一屏，点击数据表复选框。
- *在数据集名字字段里面，确定它说的是“ContactDBDataSet”并点击结束。

这些步骤把你的
新数据源和ContactDB数
据库里面的People数据
表连接起来。



向你的窗体添加数据库驱动控件

现在我们可以回到我们的窗体，并添加一些更多的控件。但是它们不仅仅是一般的控件，它们是绑定到我们的数据库和 People 数据表的列的控件。这意味着对于窗体上一个控件中的数据的改变将会自动的作用于数据库里面的相应数据列。

← 这有点费事儿，但是现在我们要回去创建和我们的数据存储交互的窗体对象。

1 选择你想用的数据源。

从数据下拉列表中选择显示数据源。这将会打开数据源窗口，显示你为你的应用设置的数据源。

如果你没看见这个标签页，选择从数据菜单里选择“显示数据源”。



↑ 这个窗口显示你的所有数据源。你仅仅设置了一个，但是你可以为更多不同的数据表和数据源设置。

你也可以找到并点击你的数据库浏览器窗口底部的数据源标签页。

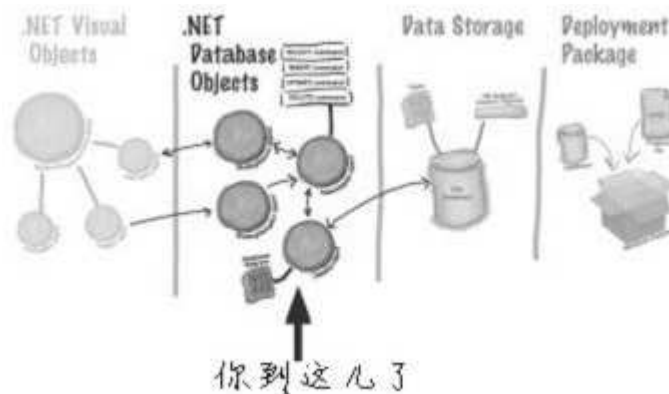
2 选择 People 数据表。

在 ContactDBDataSet 下面，你应该可以看见 People 数据表和其中所有的数据列。如果你没看见，点击下拉菜单的箭头，并且选择 Details。



这个是你应该点击的小箭头。

← 你创建的所有字段都应该显示在这儿。



3 创建绑定到 People 数据表的控件。

拖拽 People 数据表到你的窗体。你应该可以看见显示数据库中每一列的控件。不用太担心它们现在看起来的样子；就确定它们都出现在窗体上了。

如果你意外的点击到了当前窗体外面，你总是可以点击“Form1.cs[Design]”标签页，或者从解决方案浏览器打开Form1.cs来回到该窗体。

IDE创建了这个工具栏来在People数据表中导航。

这些不会出现在你的窗体上，但是它们代表数据集去和IDE创建的People数据表和ContactDB数据库交互。

当你拖拽People数据表到窗体，会为一个数据列创建一个控件。

这个对象连接你的窗体和People数据表。

这个适配器允许你的控件与IDE和数据源给你生成的SQL Command交互。

这个绑定的导航器把工具栏和你的数据表连接起来。

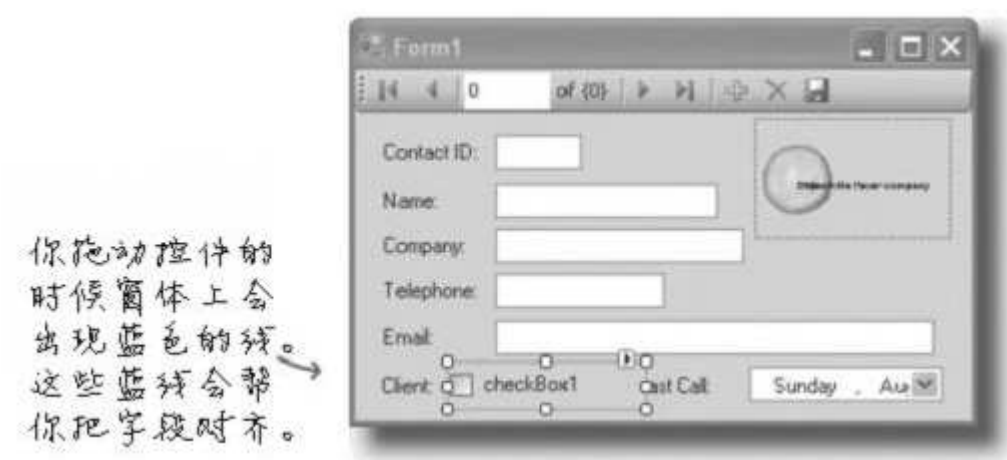
好的程序用起来是直观的

现在，窗体可以运行了。但是它不怎么好看。你的程序不仅仅是实现功能就行的。它应该易用。只需要几步你就可以把窗体弄得看起来更像本章开头的纸质卡片。



1 把你的字段和标签对齐。

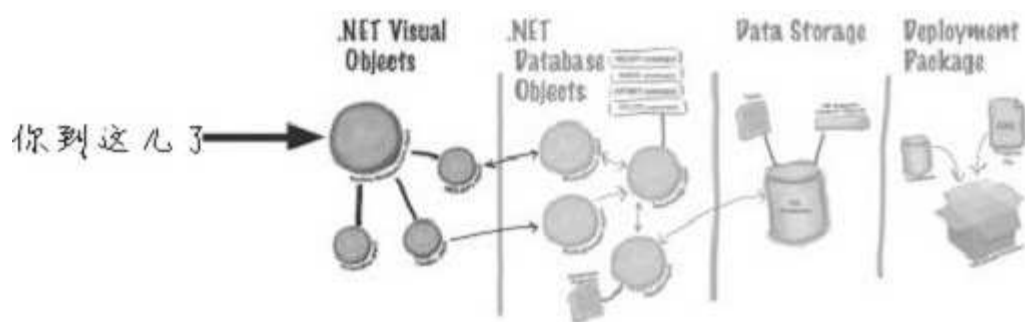
把你的字段和标签靠着窗体的左边对齐。你的窗体会看起来像其他的应用，并且使得你的用户感觉用起来更舒服。



2 改变 Client 复选框的 Text 属性。

当你第一次向窗体拖拽字段的时候你的 Client 复选框在右侧会有稍后需要删掉的一个标签。在解决方案浏览器下面可以看见属性窗口。向下滚动到 Text 属性并且删掉“checkbox1”标签。





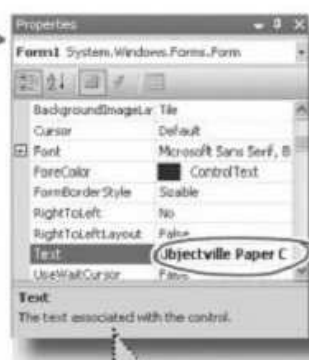
3 让应用看起来专业

你可以通过点击窗体上的任何地方并在 IDE 的属性窗口里找到 Text 属性来改变窗体的名字。把窗体的名字改变为 “Objectville Paper Co.-Contactt List”。

你也可以在同一个窗口找到 MaximizeBox 和 MinimizeBox 并把它们设置为 False 来关掉最大化 and 最小化按钮。

因为最大化窗口时
不会改变控件的位置
而使得它看起来很怪
，所以你会想要把最
大化按钮关掉。

属性窗口应该
在解决方案浏
览器下面，在
IDE 右下方。



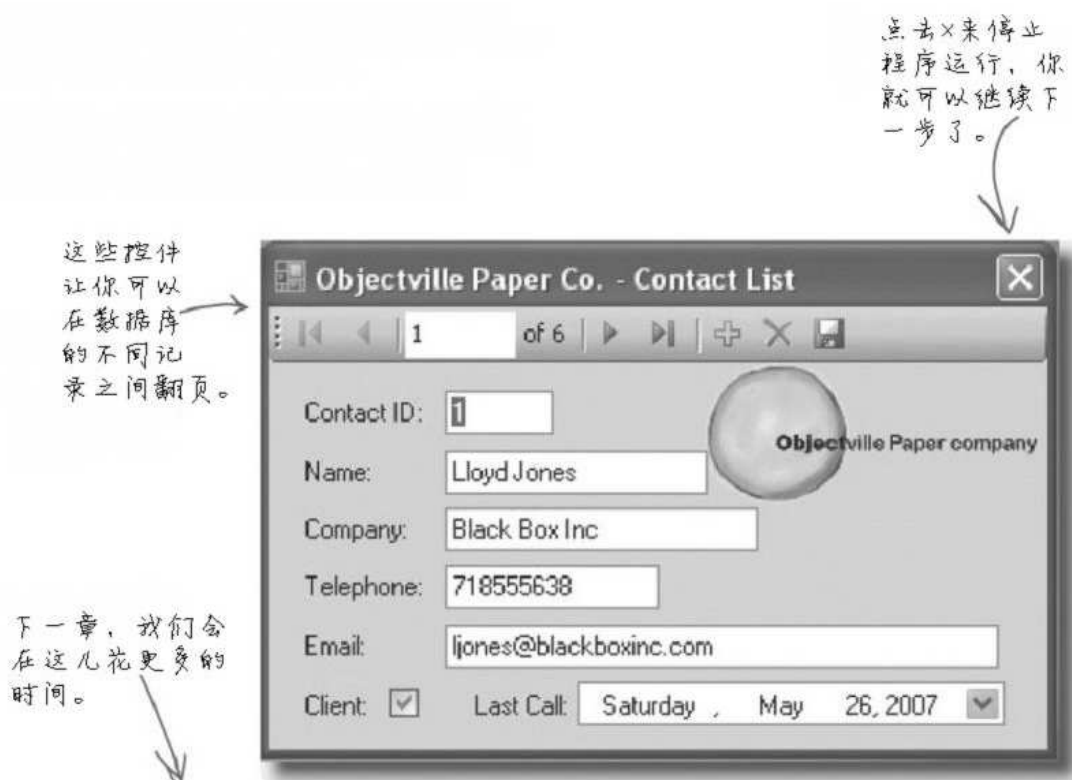
Text 属性控
制窗体标题
栏上的标题。

如果你没有属性窗口，你可以通过从
视图下拉菜单里选择它来打开它。

好的应用不仅仅要可以工作，还要易用。确定程序的行为符合标准用户的预期总会是一个好主意。

试运行

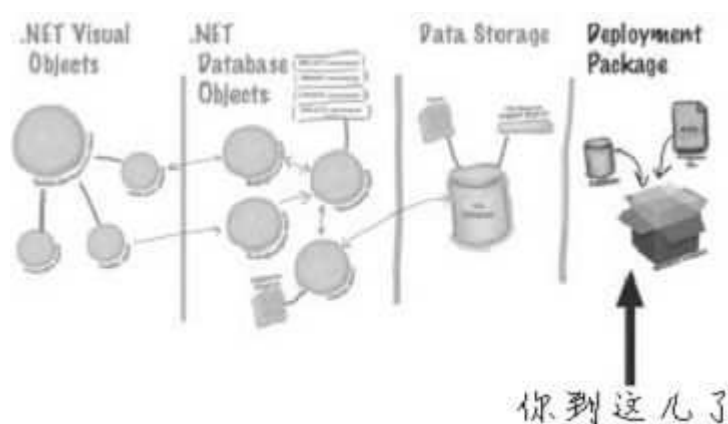
OK，只剩下一件事儿了...运行你的程序并确定它运行的和你想的一样！像原来一样来做--按键盘上的 F5 键，或者点击工具栏上的箭头一样的绿按钮（或者选择调试菜单里的“运行”）。你总是可以随时运行你的程序，甚至是还没完成的时候--尽管有可能会有错误，IDE 会告诉你并组织你运行它。



IDE 先构建，后运行。

当你在 IDE 中运行你的程序时，它实际做了两件事儿。首先构建你的程序，然后运行它。这包含了一些不同的部分。它编译代码，或者说把它转入一个可执行文件。然后他把编译过的代码和资源文件及其他文件一起放进 bin 文件夹下的一个子文件夹。

构建你的程序来覆写数据库里面的数据。



让你的应用成为每个人的应用

在这点上，你已经有一个很棒的程序。但是它只在你的机器上运行。这意味着没有别的人会

用你的应用，会付钱给你，会看见你多棒并雇用你...并且你的老板和客户看不见你从数据库生成的报告。

C#使得你部署应用变得容易。部署就是把一个应用安装到其他的机器上。有了 IDE，只需要两个步骤就可以设置好部署。

1 在生成菜单里选择发布。



2 接受发布向导的默认设置并点击完成就可以了。你将会看见它打包你的应用并展示给你一个包含你的 Setup.exe 的文件夹。



把应用给你的用户

一旦创建了一个部署文件，你将会有一个叫做 publish/的文件夹。文件夹里有一些东西，都是用来安装的。对用户来说最重要的是安装文件，安装文件使用户可以把你的程序安装到他们自己的电脑上。

所有的支持文件都存在这儿。



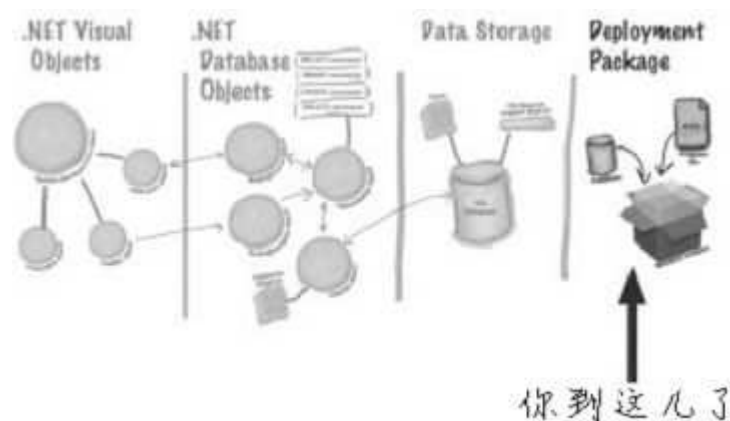
当程序被安装的时候，这个文件告诉安装包需要包含的一切。

你的用户就是用它来安装程序到他们的电脑上的。

秘书告诉我你已经让新的联系人数据库运行起来了。收拾收拾东西——我们去白杨林的直升机上为你这样的好手准备了一个座位！



老板听起来很高兴。干得好！虽然如此，在你坐上直升机去滑雪之前你还有一件事儿...



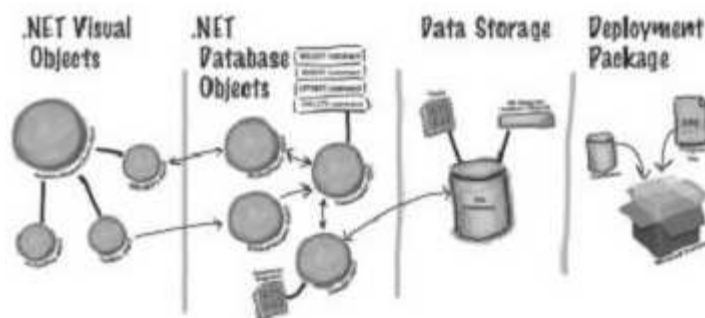
你还没完事儿：测试你的应用

在你开香槟之前，你需要测试你的部署和安装文件。在运行你的程序之前你不会把它给任何人，对吧？

关闭 VS IDE。点击安装程序，并在你的电脑上选择一个位置来安装程序。现在在那儿运行它，并且确定它运行的和你预期的一样。你也可以添加、改变记录，它们会被保存到数据库。

测试所有的东西！

测试程序，测试部署，测试程序的数据。



你已经创建了一个数据驱动的应用

VS IDE 使得创建窗体应用、创建并设计数据库和把这两项连接起来都很容易。你甚至只需要多点几下鼠标就可以生成一个安装包。



简直飞快

C#的强大就是你可以很快的准备和运行，然后集中注意于你的程序被预期要做的事情上...无需很多窗口，按钮，和 **SQL** 访问代码。

Head First C# 中文版 第二章

2都只是代码

知其所以然



你是个程序员，不只是个 **IDE** 使用者。

用 **IDE** 你可以完成很多任务。但是它也只能带你走那么远了。当然，创建一个应用的时候你有很多的**重复任务**要做。**IDE** 很善于为你做那些事儿。但是用 **IDE** 工作只是一个开始。你可以让你的程序做的更多--写 **C#代码**是实现的方式。一旦你掌握了编码的窍门，就没有你的程序做不到的了。

当你在做这个的时候...

这个 IDE 是一个强大的工具--但是它也就只是给你用的工具。每次你在 IDE 里修改你的项目或拖拽一些东西，它自动的创建代码。它很善于写样板代码，或者是不需要很多自定义的性的易复用的代码。

我们来看看在典型的应用开发中做什么，当你要...

1 创建一个新的窗口应用解决方案

IDE 允许你创建很多种应用，但是现在咱们就专注于窗口应用。窗口应用是那些很有类似窗体、按钮的程序。

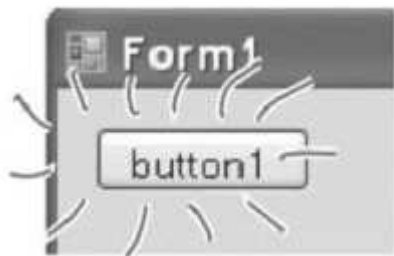
所有这些任务都要用到标准动作和样板代码。这些事是IDE善于帮你做的。

确定你总是创建一个窗口应用项目--那告诉IDE去创建一个空窗体并把它添加到你的项目中去。



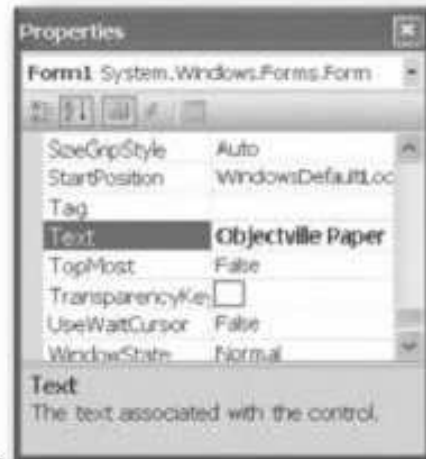
2 从工具箱拖动一个按钮去窗体上，并双击它。

按钮是你在窗体中驱动时间的方式。我们将会用很多按钮来探索 C#语言的不同部分。按钮也会是你将写的绝大多数 C#程序的组成部分。



3 设置你的窗体的一个属性

IDE 中的属性窗口确实是个强大工具，你可以用它来改变你程序中的任何元素的性质：控件的所有可视化和功能化属性，数据库的性质，甚至你的项目自身的选项。



IDE中的属性窗口只是一个
自动修改Form1.Designer.cs
中大块代码的简单途径。手
动去做要花很久时间。

...IDE 做这些

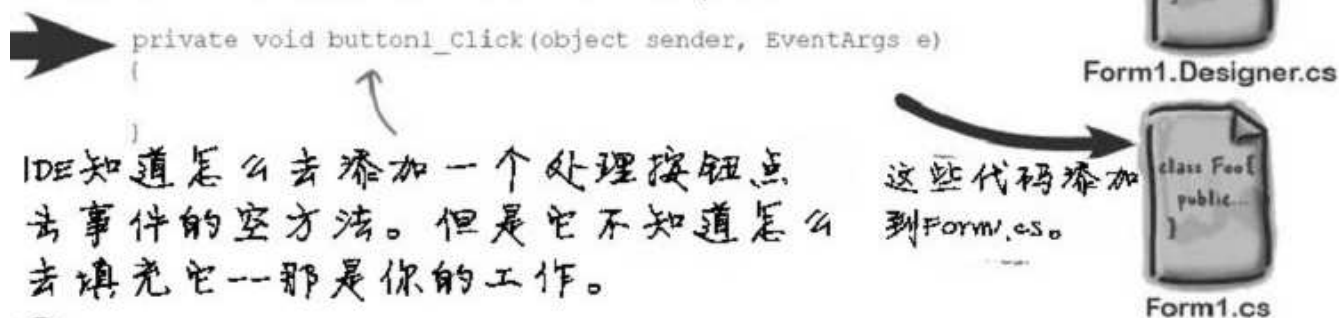
每次你在 IDE 里做些改变，它就会改变代码，这意味着改变包含代码的文件。有时只是改变几行，但有时也会添加文件到项目中去。

① ...IDE 为项目创建了
这些文件和文件夹。

这些文件是从包含
创建和显示窗体的
基本代码的预定义
模板创建来的。



- ②...IDE向Form1.Designer.cs文件添加代码
来向窗体添加按钮，然后向Form1.cs文件
添加代码来处理按钮点击事件。



- ③...IDE打开Form1.cs文件，更新一行代码。



程序从哪儿来

一个 C#程序可能从一堆文件中的声明开始，最终成了你电脑上运行的程序。下面是它怎么来的。

每个程序都从源代码文件来

你已经看见怎么编辑程序，和 IDE 怎么把你的程序保存到一个文件夹里。这些文件就是你的程序--你可以把它们复制到一个新文件夹并打开，所有东西都还在那儿：窗体，资源，代码，和其他任何你添加到你的项目的东西。

你可以认为 IDE 就是一个很帅的编辑器。它为你自动缩进，改变关键字的颜色，为你匹配括号，甚至建议下一个可能用的词。但是最后，IDE 做的所有事儿就是编辑包含你的程序的文件。



你完全可以用记事本来创建你的程序，但是那会很耗时间。

IDE 创建一个 solution (.sln) 文件和一个包含所有程序用到的文件的文件夹，以此来把程序的所有文件都绑定到这个解决方案。解决方案文件包含一个项目文件（以.csproj 结尾）列表，而项目文件包含了所有与项目关联的文件的列表。

这本书里，你只创建单项目解决方案，但是用解决方案浏览器可以很容易的添加其他项目到解决方案里。

.NET Framework 给你合适的工具

C#只是一门语言--单靠它自己，它不能做所有事。所以在这儿就需要.NET Framework 了。还记得你从窗体上去掉的最大化按钮吗？当你点击窗口上的最大化按钮的时候，有告诉窗口怎么最大化它自己并占据全屏的代码。这些代码是.NET Framework 的一部分。按钮，复选框，列表...这些都是.NET Framework 的组成部分。把你的窗体和数据库连接起来的也是一些内部代码块。它有绘制图形，读写文件，管理集合的工具，做程序员的各种日常工作的工具。



.NET Framework 中的工具分割到命名空间里。你已经见过那些命名空间了，就在你的代码最上面的“using”那些行。有一个命名空间叫做 System.Windows.Forms--你的按钮，复选框，窗体都是在那儿来的。无论何时你创建一个窗体应用项目，IDE 都会添加必要的文件以使得你的项目包含一个窗体，并且文件里最上面都有一行“using System.Windows.Forms”。

生成程序来创建可执行文件

当你选择生成菜单里的“生成解决方案”时，IDE 就会编译你的程序。IDE 通过运行编译器来把源文件生成为可执行文件。可执行文件就是双击可执行的.exe 结尾的文件。当你生成程序时，它就会在 bin 文件夹下生成可执行文件。但你发布解决方案的时候，它会把你的可执行文件及其他必要文件一起复制到你指定的发布文件夹。



你的程序在 CLR 中运行

当你双击可执行文件时，Windows 运行它，但是在你的程序和 Windows 之间还有一个特殊的“层”，叫做 Common Language Runtime，或叫做 CLR。不久之前（但是在 C# 出现之前），写程序会更难一些，因为你要自己接触硬件和底层的东西。你从不知道人们怎么配置电脑。CLR--常被称作虚拟机--通过在你的电脑和你的程序之间做“翻译”工作来为你处理那些配置问题。



你现在不需要太多的关心 CLR。知道它为你处理程序运行的问题就够了。继续前进，你会学到 CLR 的更多知识。

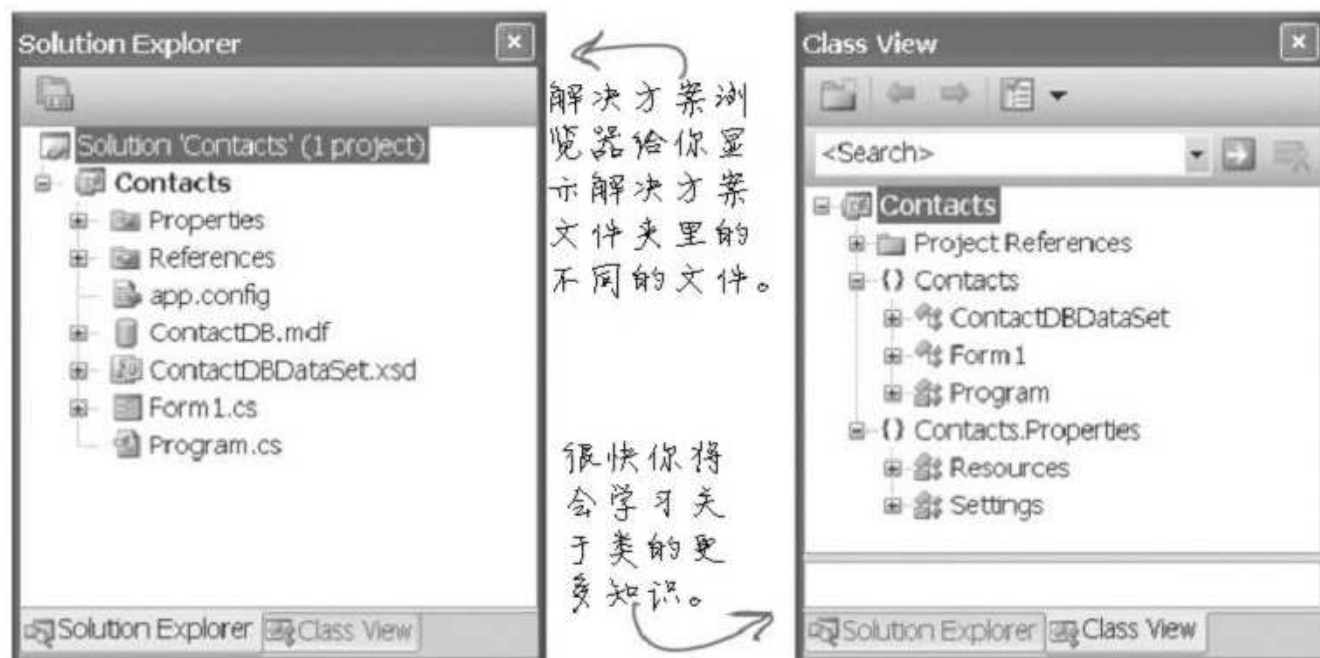
你将会了解关于 CLR 为你做的的很多事儿。比如说，它为你管理内存，在程序结束时销毁无用的数据。程序员原来要自己处理这些，现在你不用麻烦了。现在你可能不懂，但是 CLR 将会使你学习 C# 变得容易多了。

IDE 帮你编码

你已经看见了 IDE 可以做的一些事情。让我们近距离看看它给你的一些工具。

*解决方案浏览器给你显示项目中的所有东西。

你将在类之间来回切换，最简单的方式就是用解决方案浏览器。它有两种视图：解决方案视图（显示项目中的文件）和类视图（显示你的代码是怎么在逻辑上分割到不同的类）。



*使用标签页在打开的文件之间切换

因为你的程序分散在多个文件中，总会有多个文件同时打开。每个打开的文件都会在代码编辑器中处于自己的标签页中。IDE 在还没被保存的文件名前显示一个星号 (*)。

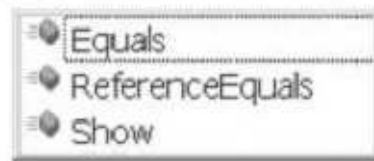


*IDE 帮你写代码

输入代码的时候你注意到弹出的小窗口了没？那是被叫做智能感应的特性，而且它真的很有用。它做的一件事儿就是为你显示这一行接下来可能完成的方式。如果你输入 `MessageBox`

然后一个点，它就知道接下来有三种方式来完成这一行。

MessageBox.



IDE知道MessageBox有三个方法叫做Equals, ReferenceEquals, 和Show。如果输入s, 它选中Show。按tab或回车来选择它, 如果你要输入很长的方法名这真的很节省时间。

如果你选择 Show 并键入 (, IDE 的智能感应将会给你显示关于怎么完成这一行的信息:

这意味着
MessageBox
的Show方法
有2种调用
方法(比如
怎么显示按
钮和图标)。

MessageBox.Show(

3 of 21 DialogResult MessageBox.Show (string text, string caption)
text: The text to display in the message box.

IDE 还有一个叫做代码片段的捷径, 它使你输入一个缩写来告诉 IDE 怎么填充剩下的代码。
有一个有用的: 输入 mbox 并按 tab 两次, IDE 将会为你填充进 MessageBox.Show:

MessageBox.Show("Test");

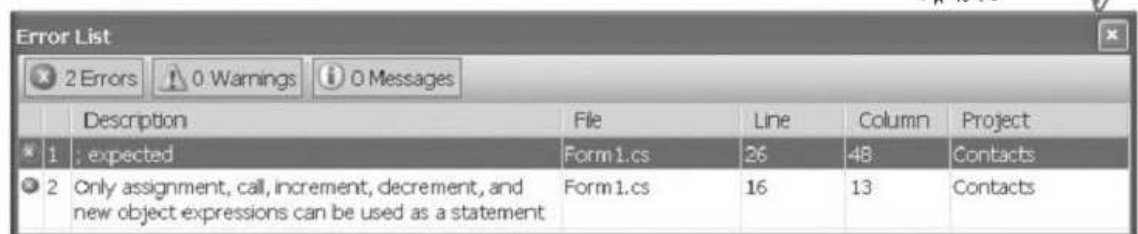
*错误列表帮你找到编译错误

如果你还没发现在 C# 程序里面写代码有多简单, 你马上就会发现了! 很幸运, IDE 给你一个很好的找错误的工具。当你生成你的解决方案时, 任何阻碍编译的问题都会显示在 IDE 底部的错误列表窗口里面。

错误列表帮你找到编译错误

如果你还没发现在 C# 程序里面写代码有多简单, 你马上就会发现了! 很幸运, IDE 给你一个很好的找错误的工具。当你生成你的解决方案时, 任何阻碍编译的问题都会显示在 IDE 底部的错误列表窗口里面。

你用开始调试在 IDE 里面运行程序的时候, IDE 首先做的就是生成程序。如果编译通过, 就会运行。不通过, 不会运行, 会在错误列表里面列出错误。



双击一个错误, 将会跳到代码中的错误处。

```
private void pictureBox1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Contact List 1.0")
}
```

你去掉分号的地方 IDE 将会显示一个红色的下划线。

在 IDE 里面改变东西就同步的改变了代码

IDE 长于为你写可视化代码。但是别全靠它。打开 VS，创建一个新窗体应用项目，亲自看看吧。

当你看见“Do this! ”，打开 IDE 并跟着做。我们将会告诉你做什么，并给你指出怎么在例子中得益最多。



1 打开设计器代码

在 IDE 里打开 Form1.Designer.cs 文件。但是这次，不是在窗体设计器里面打开，而是通过在解决方案浏览器里右击它并选择“查看代码”来打开它的代码。找到 Form1 类的声明：

```
public partial class Form1 : Form
```

注意，它是 partial class?

2 打开窗体设计器并向窗体添加一个 PictureBox

要习惯于操作多个标签页。打开 Form1.cs 的窗体设计器。拖拽一个新 PictureBox 到窗体上去。

3 找到并扩展 PictureBox 控件的设计器生成代码

回到 Form1.Designer.cs 标签页。向下滚动找到这一行代码：

点击这个加号

```
Windows Form Designer generated code
```

点击左侧的+来展开代码。向下滚动找到这些代码：

```
//
// pictureBox1
//
this.pictureBox1.Location = new System.Drawing.Point(276, 28);
this.pictureBox1.Name = "pictureBox1";
this.pictureBox1.Size = new System.Drawing.Size(100, 50);
this.pictureBox1.TabIndex = 1;
this.pictureBox1.TabStop = false;
```

如果你的代码里面关于位置、大小的代码与这儿不同，别担心……

等等！他们说什么？

倒回去一会。就在这儿，窗体设计器最上面顶儿上那一块。

```
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
```

大多数注释由两个斜线< // >开始。但是有时候IDE会添加三个斜线的注释。

对于小孩来说，没有什么比一个写着“别碰这个！”的提示更有吸引力的了。来吧，你已经被吸引了...我们来用代码编辑器修改那个方法的注释！向窗体添加一个按钮，然后做下面的：

① 修改设置button.Text属性的代码。你认为这回怎么反映到IDE中的属性窗口上呢？

试一下——看看会怎样！现在回到窗体设计器并检查Text属性，它变了吗？

② 在设计器里用属性窗口来改变别的东西的Name属性。

看你能不能用IDE来改变Name属性。它就在属性窗口的顶儿上，在“(Name)”下面。代码怎么了？代码里的注释呢？

③ 修改设置Location属性为(0, 0)的代码并

修改Size属性以使按钮变得很大。

管用么？

④ 回到设计器，并把按钮的BackColor

属性设置为别的颜色。

仔细看看代码。是不是添加了几行？

无需保存窗体或运行程序来查看改变。就在代码编辑器里面修改，回到设计器一改变效果马上就能看见。

用 **IDE** 去修改窗体设计器生成的代码总是会更简单一些。在 **IDE** 里面做的任何修改最终都会反映到项目代码的改变上。

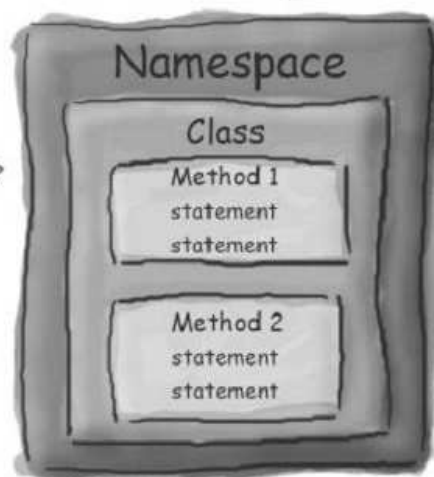
剖析一个程序

每个 C# 程序的代码都是以相同的方式组织的。所有程序都用命名空间，类，和方法来使你的代码易于管理。

每次创建一个新程序，都会给它定义一个命名空间，这样它的类就和 .NET Framework 的类区分开来。

一个类包含你的程序的一部分（尽管有的程序只有一个类）。

一个类有一个或多个方法。你的方法也必须处于类中。方法是由语句组成的——就像你已经看见的那样。



咱们近距离看看你的代码

打开你的通讯簿项目的 Form1.cs，我们来一点一点看看。

1 代码文件以引用.NET Framework 的工具开始

在每个代码开头都可以看见一堆写 using 的行。他们告诉 C# 要用 .NET Framework 的哪部分。

如果你要用其他命名空间的类，自己添加 using 来引用它们。因为窗体经常用一些 .NET Framework 里的工具，IDE 会自动添加它们。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

这些 using 行处于每个代码文件的开头。它们告诉 C# 用哪些 .NET Framework 的类。每一行告诉你的程序引用 .NET Framework 一些特定类。

2 C# 程序被组织到类里

每个 C# 程序都被组织到类里。一个类可以做任何事儿，但是大多数类只做一件特定的事儿。

创建这个程序的时候，IDE 添加一个叫做 Form1 的类来显示窗体。

```
namespace Contacts
{
    public partial class Form1 : Form
    {

```

你给程序命名为Contact，IDE就在代码最顶部添加一个叫做Contact的命名空间。这一对大括号里的任何东西都是Contact命名空间的一部分。

这个类叫做Form1。它包含所有的绘制窗体的代码和Toolbox的代码。你告诉IDE创建窗体应用时，IDE就会添加这些代码。

3 类包含执行动作的方法。

类有事儿要做时，就用到了方法。方法接受输入，做些处理，有时还会有输出。给方法输入要通过参数。方法的不同行为取决于不同的参数。有的方法产出输出。这时，输出就叫做返回值。如果方法前有 void 关键字，说明没有返回值。

```
public Form1()
{
    InitializeComponent();
}
```

这行调用一个叫做InitializeComponent()的方法，它也是IDE给你创建的。

4 一个指令执行一个动作

当你添加 MessageBox.Show() 这一行到程序里，就是写了一个指令。每个方法都是由指令组成的。程序调用一个方法，先执行第一个指令，然后下一个，下一个，等等。当方法执行完了指令或者遇到了一个 return 指令，它就停止，然后程序返回到调用处。

```
private void pictureBox1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Contact List 1.0", "About");
}
```

这个方法叫做pictureBox1_Click()，用户点击PictureBox时就会调用它。

这个方法有两个参数，叫做sender和e。

这就是一个指令。你已经知道它做什么了——它弹出一个小消息框窗口。

你的指令调用这个Show()方法，它是MessageBox类的一部分，它处于System.Windows.Forms命名空间。

你的指令传给Show()方法两个参数。第一个是在消息框窗口中显示的文字，第二个是用来在消息框窗口的标题栏上显示的。

你的程序知道从哪儿开始

创建一个窗体应用，IDE添加的文件里面有一个叫做Program.cs。去解决方案浏览器里双击它。它里面有一个类叫Program，类里面有一个叫做Main()的方法。这个方法就是程序入口点，也就是说这个方法是程序里第一个被执行的。

每个C#程序只能有一个入口点方法，这个方法总是叫做Main()。这样，运行程序时程序就知道从哪儿开始了。

下面的代码是上一章IDE给你自动创建的。在Program.cs里面可以找到它。

靠近你的代码

```

1 using System;
  using System.Linq;
  using System.Collections.Generic;
  using System.Windows.Forms;

2 namespace Contacts
  {
3     static class Program
4     {
5         /// <summary>
6         /// The main entry point for the application.
7         /// </summary>
8         [STAThread]
9         static void Main()
10        {
11            Application.EnableVisualStyles();
12            Application.SetCompatibleTextRenderingDefault(false);
13            Application.Run(new Form1());
14        }
15    }
16 }

```

这儿的代码都处于Contact命名空间里。下面几页我们将更多的讨论命名空间。

斜线开始的行是注释，你可以在任何地方添加注释。斜线告诉C#忽略它们所在行。

每次你运行程序，它在这儿开始，就是这个入口点。

这条指令创建并显示联系人窗口，并在窗口关闭时结束程序。

我做声明!

每一个类或方法开头的第一行叫做声明。

记住，这只是你深入研究代码的一个开始。但是在你开始深究之前，你需要知道你在看的是什么。

1 C#和.NET 有很多内建特性

几乎每个C#类文件顶部都可以找到上页类似的代码行。System.Windows.Forms 是一个命名

空间。using System.Windows.Forms 这一行使得这个命名空间内的一切在你的程序中都是可用的。在我们的情况下，这个命名空间有很多可视化元素，比如按钮和窗体。

← 随着你用这本书学习C#和.NET的其他内建特性，你的程序将会用到越来越多的命名空间，就像这个一样。

2 IDE 给你的代码选择一个命名空间

IDE 给你创建命名空间--它基于项目名字选择 Contacts 作为命名空间的名字。你的程序的代码都处于这个命名空间里面。

命名空间让你可以在不同的程序中使用相同的名字，只要这些程序不处于同一个命名空间。

3 你的代码存在类中

这个类叫做 Program。IDE 创建它，并添加了开始程序和显示窗体的代码进去。

← 一个命名空间里面可以有多个类。

4 这段代码有一个方法，内有三条指令。

命名空间里有类，类里有方法。每个方法里又有一些指令。在这个程序内，这些指令处理开始 Contacts 窗体的事儿。动作都在方法内发生--每个方法都做些事儿。

5 每个程序都有一个叫做入口点的特殊方法。

每个 C#程序肯定有一个叫做 Main 的方法。即使你的程序有许多方法，只有一个可以被首先执行，那就是 Main 方法。C#在你的代码里查找一个标有 static void Main () 的方法。然后，程序执行时，Main 方法的第一条指令被执行，所有其他东西都跟随而来。

每个 **C#**程序肯定有一个叫做 **Main** 的方法。

这个方法是你的代码的入口点。执行时，**Main** () 方法里的代码首先被执行。

你可以改变程序的入口点

既然你的程序有一个入口点，它在哪个类里、它做什么就无所谓了。我们删掉 Program.cs 里面的 Main 方法，并自己创建一个新的。



- ① 回到Program.cs并把Main方法更名为NotMain。
现在试着生成并运行程序。发生什么了？

写下改变了方法名
发生了什么，
并写下你认为
什么会发生。

- ② 现在创建新入口点。添加一个叫做AnotherClass.cs的
类。在解决方案浏览器里右击项目名，选择“添加>>
类...”。把类命名为AnotherClass.cs。IDE就会向项
目中添加一个叫做AnotherClass.cs的类。下面就是这个类：

右击项目
名，并选
择添加类。

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;

namespace Contacts
{
    class AnotherClass
    {
    }
}
```

这四个标准的using行
被添加到这个文件

这个类处于你最初创建窗体
应用时IDE添加的Contacts
命名空间中。

IDE根据文件名
自动给类命名。

- 3 给文件顶部添加一个 **using** 行： **using System.Windows.Forms;** 别忘了用分号结尾！

- 4 通过向大括号中键入下面代码来向 **AnotherClass** 类添加这个方法。

MessageBox是System.Windows
.Forms命名空间里的一个类，
所以你必须添加第三步骤
中的using行。Show()方法
是MessageBox类的一部分。

```
class AnotherClass
{
    public static void Main()
    {
        MessageBox.Show("Pow!");
    }
}
```



怎么样了？

现在你的程序不会显示联系人窗体了，它只弹出消息框窗口。你写了新的 `Main()` 方法，你就给了程序一个新的入口点。现在程序做的第一件事就是运行 `Main` 里的指令--也就是运行 `MessageBox.Show()` 这条指令。`Main` 里除了这条就没有别的指令了，所以你点击 `OK` 按钮之后，程序没指令可执行了，它将会停止。

5 想想怎么修复你的程序，让它还可以弹出联系人窗口。

提示：你只需要修改两个文件里的两行代码就可以了。

there are no
Dumb Questions

问：大括号都是干嘛用的？

答：C#用大括号来把指令组织到代码块里。大括号总是成对的。一个左括号后总是可以看见一个右括号。IDE 帮你匹配大括号--单击一个括号，它的配对括号和它将会被阴影显示。

问：我不太懂入口点是什么。可以再给我解释一下嘛？

答：程序有很多指令。但是它们不是一起执行的。程序从第一个指令开始，然后下一个，下一个，再下一个，等等。这些指令通常组织进类里，那程序怎么知道从哪儿开始呢？

入口点就是这么来的。除非有一个叫做 `Main()` 方法的入口点，否则编译器无法编译你的代码。程序从 `Main()` 方法里的第一个指令开始执行。

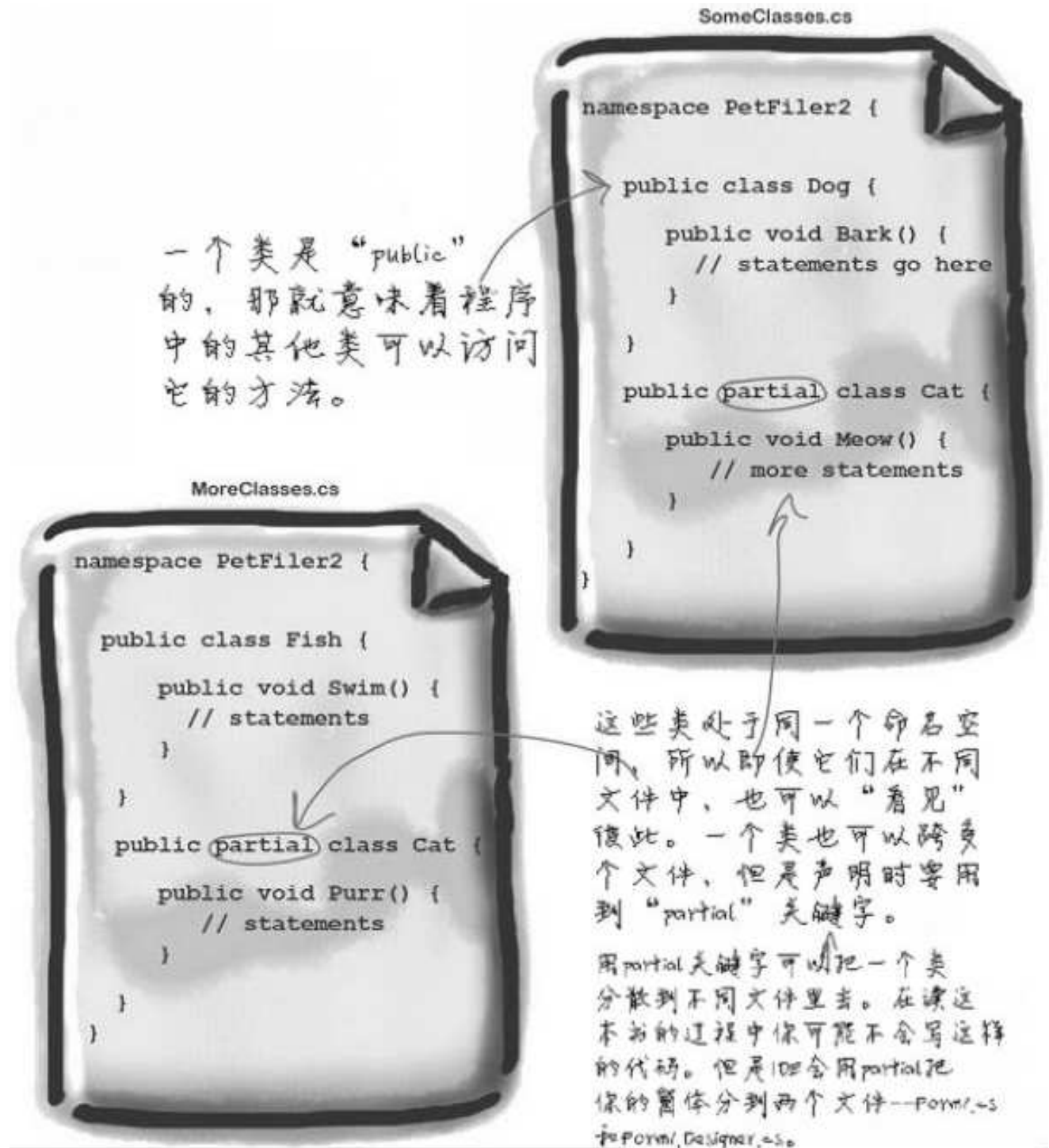
问：怎么在我运行的时候在错误列表里出现错误提示了呢？我认为只有“生成解决方案”是才会那样。

答：因为你运行程序时，发生的第一件事就是保存解决方案里的所有文件并尝试去编译。而编译的时候--不管是运行是还是生成时--如果有错误，IDE 都会把错误显示在错误列表中而不会去运行程序。

前面几页是连线游戏，跳过

两个类可以处于同一个命名空间

看看这个 PetFiler2 程序中的两个类文件。两个类文件内有三个类：Dog，Cat，和 Fish。因为它们处于同一个命名空间，所以 Dog.Bark() 方法里可以调用 Cat.Meow() 和 Fish.Swim()。这些命名空间和类分散在不同文件中并不要紧，运行起来还是一样的。

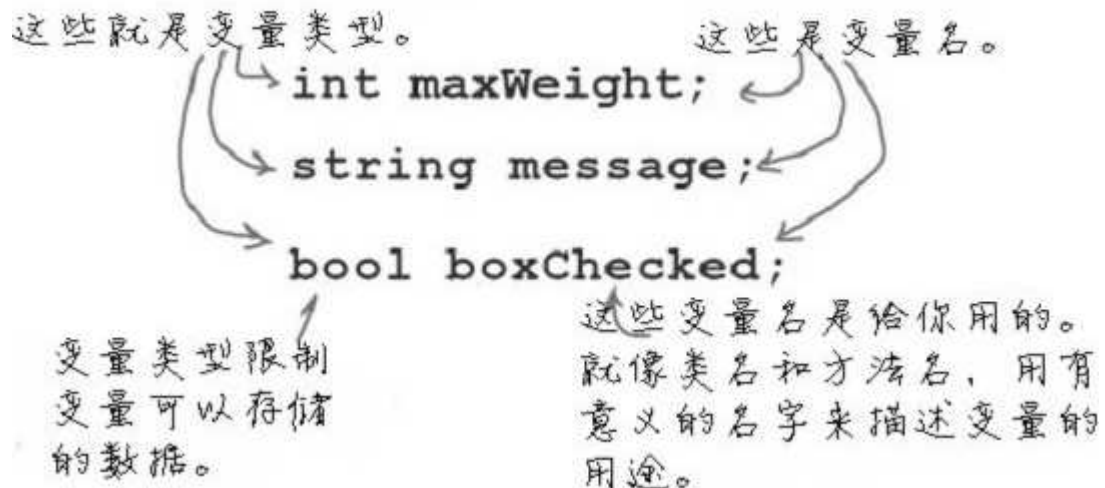


你的程序用变量来操作数据

彻底了解后，你会知道每个程序都是处理数据的。数据有时会是文件中的表单，有时是游戏里的图像，有时是一条即时消息。这些都是数据。变量此时就有用了。程序用变量存储数据。

声明你的数据

声明一个变量，你要告诉程序它的类型和名字。一旦 C#知道了变量的类型，它将会在你犯了错误或作一些没有道理的事儿时组织编译，比如用“Fido”去减 48353.



变量可变

程序运行过程中变量会有不同的值。换句话说，变量的值可变。（所以叫“变量”是个好名字。）这个很重要，因为这是你写的和将要写的程序的核心思想。所以如果你的程序把 `myHeight` 变量设置为 63；

```
Int myHeight=63;
```

每次 `myHeight` 出现在代码里，C#将会用它的值 63 来代替。然后，如果你把它改作 12：

```
myHeight=12;
```

C#将会用 12 代替 `myHeight`--但是这个变量仍叫做 `myHeight`。

无论何时你的程序需要操作数

字，文本，**True/False** 值，或

者其他任何数据，你将会用变

量来记录它们。

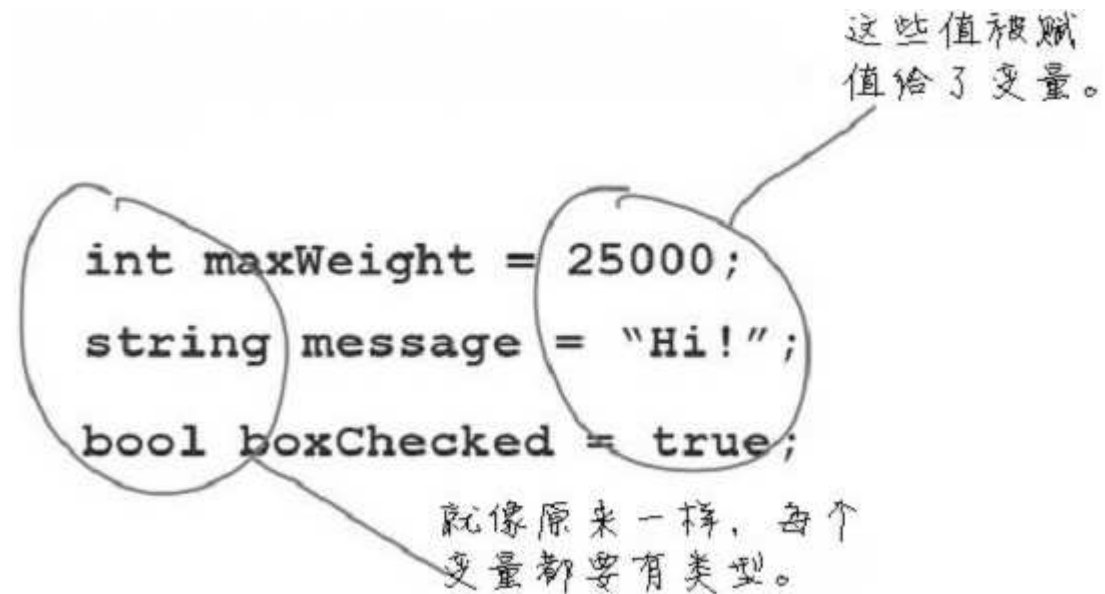
使用变量之前必须赋值

把下面的指令写入一个 C# 程序：

```
Int z;
```

```
MessageBox.Show(“The answer is” +z);
```

向前走，试一试。你会收到一个错误，IDE 将会拒绝编译代码。那是因为 IDE 将会检查每一个变量来确定它已经在被使用之前被赋值过了。防止忘记赋值的最简单方法就是把声明变量和给它赋值的指令合为一句。



一些有用的类型

每个变量都有一个类型，这告诉 C# 它可以承载什么数据。第四章我们会研究更多 C# 中的类型的细节。现在，我们将会专注于这三个最流行的类型。Int 承载整形（或者叫做整数），string 承载文本，还有 bool 承载布尔值 true/false。

如果你写的代码用了没有赋值的变量，那代码没法编译。防止忘记赋值的最简单方法就是把声明变量和给它赋值的指令合为一句。

↑
你给变量赋值，
值还可以改变。
所以在声明的
时候给变量赋
值没有什么坏处。

C# 使用常见的数学符号

你把数据存在变量里面了，你可以怎么操作它呢？好，如果它是一个数字，你可能想要对它

加减乘除。这就用到操作符了。你已经知道几个基本的了。我们来多讨论几个。下面是一段代码，它用操作符做一些简单的数学运算：

我们定义一个新int变量叫做number，并赋值15。然后给它加15。第二句指令之后，number等于25。

第三句改变number的值，把它设为36乘15，等于540。然后再重设值为12 - (42/7)，也就是6了。

+=与+=类似，不过它是把现在的值乘以3，所以最后它是48。

这个消息窗口将会弹出“hello again hello”。

“”是一个空字符串。

一个bool存储true或false值。！运算符是NOT的意思，它把true转化为false，反之亦然。

```

int number = 15;
number = number + 10;
number = 36 * 15;
number = 12 - (42 / 7);
number += 10;
number *= 3;
number = 71 / 3;

int count = 0;
count ++;
count --;

string result = "hello";
result += " again " + result;
MessageBox.Show(result);
result = "the value is: " + count;
result = "";

bool yesNo = false;
bool anotherBool = true;
yesNo = !anotherBool;

```

71除以3是23.666666。因为number是一个整型，它只可以存整数，所以它等于23。

你将会经常用int计数，你用int的时候++和--运算符是很方便的。++给count加一，--给count减一，所以最后它还是0。

当你用字符串应用+时，它把两个字符串连到一块。它将会自动把数字转化为字符串。

不用急着去记住这些操作符。

你后面还会一遍又一遍的见到他们。



循环反复执行一个动作

对大多数比较大的程序有一个奇怪的事儿：它们经常反复调用一个特定的动作。循环就是做这个用的--它们告诉程序持续执行一段特定代码直到某些状态成为 true 或 false。

这就是布尔值如此重要的原因。循环用布尔值来判断它是否应该继续执行。

```
while (x > 5)
{
    x = x - 3;
}
```

在一个while循环内，所有大括号里的指令只要圆括号里的状态为真就会反复被执行。

每一个for循环都有三个指令。第一个设置循环。第二个表达式只要为真循环就会继续。第三条指令在每次循环之后都会得以执行。

```
for (i = 0; i < 8; i = i + 2)
{
    MessageBox.Show("I'll pop up 4 times");
}
```

用一小段代码写一个 for 循环

一分钟就能写完一个 for 循环。IDE 还能帮你加快一点。键入 for 和两次 tab，IDE 将会自动为你插入代码。如果你键入一个新变量，它将会自动更新剩下的代码段。再按一次 tab，光标将会跳到 length 上去。

如果你把这个变量改变为其他的，代码段将会自动改变另外两个。

```
for (int i = 0; i < length; i++)
```

按tab来让光标跳到length上去。循环执行的次数取决于你把length设置为多少。你也可以把length改变为一个数字或者一个变量

开始编码喽

任何程序的真正工作都在于写指令语句。但是语句不是存在于真空里的。所以我们开始进入深入挖掘并写写代码吧。创建一个新窗体应用项目。



添加指令语句来显示一条消息

从双击第一个按钮来开始。然后给 `button1_Click()` 方法添加下面的六条指令语句。近距离观察代码并看看它的输出。

x 是一个变量。“int”告诉C#它是一个整数，剩下的代码把它的值设置为3。

```
private void button1_Click(object sender, EventArgs e)
{
    // this is a comment
    String name = "Quentin";
    int x = 3;
    x = x * 17;
    double d = Math.PI / 2;
    MessageBox.Show("name is " + name
        + "\nx is " + x
        + "\nd is " + d);
}
```

这是个叫做Math的内建类，它有一个叫做PI的成员。Math存在于System命名空间里，所以这段代码所在的文件开头要有using System这一行。

\n 是一个换行的转义序列。

语法 101

*每行代码都是分号结尾

`x=x+1;`

*注释以双斜线开始

`//this line is ignored`

*大多数空格都无所谓

`x = 3;` 与 `x=3;` 一样

*变量声明时都有类型和名字（第四章你会学习很多的类型）

`int weight;`


```
//weight is an integer
*类和方法定义时都有大括号
public void go()
{
    //amazing code here
}
```

If/else 语句决定走向

用 if/else 语句告诉程序在你设定的状态为真（或为假）的时候去做特定的事情。很多 if/else 语句检查两个表达式是否相等。这时你就要用到 == 运算符了。这个和你用来赋值的 = 运算符不同。

```
if (someValue == 24)
{
    MessageBox.Show("The value was 24.");
}
```

每一个if语句都以状态检查来开始。

这句大括号里面的语句只有当测试为真的时候才执行。

我们经常用双等号来检查两个表达式是否相等。

```
if (someValue == 24)
{
    // You can have as many statements
    // as you want inside the brackets
    MessageBox.Show("The value was 24.");
} else {
    MessageBox.Show("The value wasn't 24.");
}
```

if/else语句是很直率的。如果状态为真则执行第一对大括号里面的内容，否则就执行第二对大括号里面的语句。



别被双等运算符弄糊涂了！

你用一个等号 (=) 来给变量赋值，但是两个等号 (==) 是用来比较两个变量的。你无法相信程序中的多少 bug--即使是老手程序员写的！--是因为错把 == 用成了 =。如果你看见 IDE 提示你“不能隐式把 int 转化为 bool”，那可能就是出上面讲的差错了。

设置条件并看看是不是真的

用 if/else 语句告诉你的程序当条件为真（或假）的时候去做特定的事情。

用逻辑运算符来检查条件

你已经看过 == 运算符了，你用它来检查两个变量是不是相等。还有很多其他的运算符。你不用现在急着记住它们--下面几章你将会一遍又一遍的见到它们。

* != 运算符和 == 类似，只是它当在你比较的两个变量不等时才为真。

* 你可以用 < 或 > 来比较数字，来看看哪个大哪个小。

* ==, !=, <, > 叫做条件运算符。当你用它们测试两个变量或值时，这叫做执行条件测试。

* 你可以用 && 代表 AND，|| 代表 OR 来把两个条件测试合并到一个测试中。所以检查 i 是否等于 3 或者 j 是否小于 5，就这么写：(i == 3) || (j < 5)。

用条件测试运算

符来比较两个数

字叫做条件测试。

设置一个变量并检查它的值

确定你做这一步时已经停止了程序运行--因为 IDE 禁止你在程序运行的时候修改代码。

这儿是第二个按钮的代码。这是检查一个叫做 x 的整数变量是否等于 10 的指令。

```
private void button2_Click(object sender, EventArgs e)
{
    int x = 5;
    if (x == 10)
    {
        MessageBox.Show("x must be 10");
    }
    else
    {
        MessageBox.Show("x isn't 10");
    }
}
```

首先设置一个 x 整形变量为 5。然后检查它是否等于 10。



这是输出。看你能不能改一行代码来使得它显示 "x must be 10"

添加另一个条件测试

第三个按钮做下面的输出。现在改两行代码来让它弹出两个消息框窗口。



```
private void button3_Click(object sender, EventArgs e)
{
    int someValue = 4;
    String name = "Bobbo Jr.";
    if ((someValue < 3) && (name.Equals("Joe")))
    {
        MessageBox.Show("x is 3 and the name is Joe");
    }
    MessageBox.Show("this line runs no matter what");
}
```

这一行检查 someValue 不等于 3，然后检查 确定 name 值为 "Joe"。

向你的程序添加循环

这儿是最后一个按钮的代码。它有两个循环。第一个是 while 循环，它在条件为真时反复执行括号内的代码。第二个是 for 循环。看看它怎么工作的。

```
private void button4_Click(object sender, EventArgs e)
{
    int count = 0;
    while (count < 10)
    {
        count = count + 1;
    }
    for (int i = 0; i < 5; i++)
    {
        count = count - 1;
    }
    MessageBox.Show("The answer is " + count);
}
```

这个循环在 count 值小于 10 时，反复执行。

这儿设置循环。它给整形设置下面会用的一个值。

for 语句的第二部分是测试语句。它说“只要 i 小于 5，我就反复执行”。

在这儿循环真的做点事儿。在这个例子里它给 i 加一。所以每次循环都会给 i 加一。

点击按钮之前，读读代码并试着弄明白消息窗口会

显示什么。然后点击按钮来看看你的猜测正确否。



我们多练习一些条件测试和循环。看看下面的代码。圈出条件测试并填空。让注释解释正在被运行的代码。

```
int result = 0; // this variable will hold the final result 我们已经给你
int x = 6; // declare a variable x and 赋值为6. ← 填写了一个。
while (x > 3) {
    // execute these statements as long as .....
    result = result + x; // add x .....
    x = x - 1; // subtract .....
}
for (int z = 1; z < 3; z = z + 1) {
    // start the loop by .....
    // keep looping as long as .....
    // after each loop, .....
    result = result + z; // .....
}
// The next statement will pop up a message box that says
// .....
MessageBox.Show("The result is " + result);
```

条件测试的更多知识

你可以用比较运算符检查一个变量来做简单的条件测试。下面是怎么比较 **x** 和 **y** 值：

x < y (小于)
 x > y (大于)
 x == y (等于)

这些是最常用的。



那你的循环就一直运行！

每次程序运行条件测试，结果或是真或是假。如果是真，那你的程序就会多循环一次。每个循环都应该有在运行次数足够之后可以使得返回结果为假的代码。如果没有，那么这个循环就一直执行，知道你关掉程序或者关掉电脑。



Loop #1

```
int count = 5;
while (count > 0) {
    count = count * 3;
    count = count * -1;
}
```

Loop #2

```
int i = 0;
while (i == 0) {
    count = count * 3;
    count = count * -1;
}
```

下面是几个循环。写出它们会终止还是会一直循环。如果会终止，它运行多少次？

Loop #3

```
int j = 2;
for (int i = 1; i < 100;
     i = i * 2)
{
    j = j - i;
    while (j < 25)
    {
        j = j + 5;
    }
}
```

Loop #4

```
while (true) { int i = 1; }
```

Loop #5

```
int p = 2;
for (int q = 2; q < 32;
     q = q * 2)
{
    while (p < q)
    {
        p = p * 2;
    }
    q = p - q;
}
```



有什么原因会让你写一个永不退出的循环？



Sharpen your pencil Solution

我们要练习一些条件测试和循环。看看下面的代码。圈出条件测试并填空。让注释解释正在被运行的代码。

```
int result = 0; // this variable will hold the final result
int x = 6; // declare a variable x and 给它赋值为6.
while (x > 3) {
    // execute these statements as long as x大于3
    result = result + x; // add x 到result变量。
    x = x - 1; // subtract 1 from the value of x
}
// 这个循环运行3次--第一次z是1, 第二次z是2, 一旦z为3, 循环就不再继续了。
for (int z = 1; z < 3; z = z + 1) {
    // start the loop by 定义z变量, 并赋值为1
    // keep looping as long as z小于3
    // after each loop, 给z加1
    result = result + z; // 把z的值加到result
}
// The next statement will pop up a message box that says
// result是18
MessageBox.Show("The result is " + result);
```



Sharpen your pencil Solution

下面是几个循环。写出它们会一直执行还是会终止。如果会停止, 那么会运行多少次?

Loop #1
执行一次

Loop #3
执行7次

Loop #5
执行0次

Loop #2
一直执行

Loop #4
死循环

there are no
Dumb Questions

问: 所有代码都要在类里面?

答: 是的。C#程序做什么都是因为执行了指令。这些指令是类的一部分, 而类是命名空间

的一部分。即使有时候有些东西看起来不像类里的指令--比如你用设计器修改窗体上某个对象的属性--但是如果你查看代码，你会发现 IDE 在代码的某些地方添加了或者修改了一些代码。

问：有没有不许我用的命名空间？有没有我一定要用的命名空间？

答：是的，有的命名空间是不许你用的。注意到 C#代码里顶部的 using 行都写着 System 了吗？那是因为有一个 .NET Framework 的命名空间叫做 System。就在那儿你可以找到所有给你的程序添加动力的工具。比如 System.Data, 它让你可以操作数据表 and 数据库。System.Math, 它有很多数学功能。大多数情况下，你可以给命名空间命名为任何你喜欢的名字（前提是只有字母、数字、下划线）。创建程序时，IDE 将会自动根据程序名字选择命名空间的名字。

问：我还是不懂，我为什么需要 partial class 这种东西？

答：用 partial class 你可以把一个类分散到多个文件。IDE 创建一个窗体的时候--它把你编辑的代码存在一个文件里（比如 Form1.cs），并把它自动修改的代码存在另一个文件里（Form1.Designer.cs）。你不需要对命名空间做同样的事。命名空间可以分散到两个，三个，一打或者更多文件里。把命名空间定义放在文件开头，下面的大括号里的东西都属于这个命名空间。另一件事儿：一个文件里可以有多个类。一个文件里也可以由多个命名空间。下面几章将会学习更多关于类的知识。

问：假如说我拖拽了一些东西到窗体上去，那么 IDE 会自动给我添加一些代码。如果我点击“撤销”，那些代码会怎么样？

答：最好的答案就是试一试！试一下--对 IDE 生成的代码做点什么。向窗体添加一个按钮，修改属性。然后试着撤销它。怎么样？你会看见 IDE 聪明到可以撤销简单的东西。但是对于更复杂的事儿，比如向项目添加一个新的 SQL 数据库，你会收到一个警告提示。IDE 知道怎么撤销，但是可能不知道恢复了。

问：我到底应该多么小心的对待 IDE 生成的代码呢？

答：你应该很小心。知道 IDE 在做什么很有用，有时需要解决严重问题时你需要知道具体 IDE 怎么处理代码。但是在大多数情况下，用 IDE 你就可以做完你想要做的事儿了。



- *用指令告诉程序做什么，指令是类的一部分，而类是命名空间的一部分。
- *每条指令以分号（;）结尾
- *你用 VS IDE 里的可视化工具时，它自动添加或修改代码
- *代码块由大括号 {} 包围。类，while 循环，if/else 语句和很多其他语句需用这些代码块。
- *条件测试或是真，或是假。你用条件测试决定循环什么时候结束，决定 if/else 的走向哪一块代码。
- *程序需要存储数据时，就用变量。用=赋值，用==判断是否相等。
- *只要条件测试为真，while 循环就执行代码块里面的每一句。
- *如果条件测试为假，while 循环停止，程序继续执行循环后面的代码。



代码贴士

一个 C# 程序的一部分粘在了冰箱上了。你能够把这些代码片重新排序形成一个可以运行显示一个消息框窗口的程序吗？有些括号掉在地上了，它们太小了捡不起来，所以尽管添加你认为需要的代码吧！

“ ” 代表一个空字符串——那意味着 Result 里面没有字母。

string Result = "";

这儿贴士没有掉在地上...

MessageBox.Show(Result);

```
if (x == 1) {
    Result = Result + "d";
    x = x - 1;
}
```

```
if (x == 2) {
    Result = Result + "b c";
}
```

```
if (x > 2) {
    Result = Result + "a";
}
```

```
int x = 3;
```

```
x = x - 1;
Result = Result + "-";
```

```
while (x > 0) {
```

Output:



——→ 答案在 82 页

本书将会给你很多类似这样的练习。几页之内我们就会给你答案。如果你卡壳了，不要担心，去看看答案吧--不算作弊！



Exercise

该用if/else指令做点练习了。你能创建这个程序吗？

这个是窗体。

添加这个复选框。
从工具箱里把它拖拽到窗体。
用Text属性修改它旁边的文本。
(你也可以用Text属性修改按钮和标签的文本)

这个是标签。
你可以用属性修改字体大小并把它设为粗体。把BackColor属性设置为红色--从选项里选择“Red”。

如果用户点击按钮并且复选框没有勾选就弹出下面这个消息。

如果你的复选框叫做checkbox1 (如果你喜欢，你也可以修改Name属性)，下面是检查它是否勾选的条件测试：

```
checkbox1.Checked == true
```



如果用户点击按钮并且复选框勾选了，就改变标签的背景颜色。

如果标签的背景颜色是红色，点击按钮就变成蓝色。如果是蓝色，就变成红色。下面的指令设置一个叫做 label1 的标签的背景颜色：

```
Label1.BackColor=Color.Red;
```

(提示：测试标签的背景颜色是不是红色的代码和上面的很像--但是有一点小区别！)



咱们来弄点炫的！

Exercise

① 要创建这个窗体

提示：如果在for循环里面声明一个变量--for (int c=0; ...)--那么这个变量只在循环大括号内部可用。如果你有两个都用这个变量的for循环，那你需要在每一个循环里定义它或者在循环体外定义。



② 让窗体背景幻化色彩！

当点击按钮时，使得窗体背景在很多颜色间变换！创建一个循环，内有。变量，值从0到254。下面是大括号里的代码块：

```
this.BackColor = Color.FromArgb(c, 255 - c, c);
```

```
Application.DoEvents();
```

这一行告诉程序让操作系统在你的程序外面处理其他事儿。没有这一行，你的程序会占用所有CPU周期--不再监听事件（比如说点击X就关闭窗口）。

颜色印象！

.NET有很多预设颜色，比如蓝色、红色，但是它也让你可以用Color.FromArgb()方法创建自己的颜色。需要三个参数：红、绿、蓝值。

③ 让它慢一点

让闪烁的慢一点，要在Application.DoEvents()之前加这行代码：

```
System.Threading.Thread.Sleep(3);
```

这个指令在循环里延迟了3毫秒。它是.NET类库的一部分，它处于System.Threading命名空间。

看这本书你将会创建很多应用，你需要给每一个取一个不同的名字。我们建议给这个取名为“2 Fun with if-else statement”，这是基于章节号和窗体上标题栏的文本。

4 平滑一点

我们回到颜色循环开始的地方。添加另一个循环让 c 从 254 到 0 走。用那个大括号内的相同代码。

5 保持运行

用一个一直循环不停地循环把你的两个循环包起来，所以点击了按钮，颜色就不停的变换。

（提示：while（true）循环永不停止！）

一个循环在
另一个循环
里面叫做嵌
套循环。

啊！程序停不下来！

在 IDE 里运行程序。开始循环。现在关闭窗口。等一会--IDE 不会回到编辑模式！它表现的就像程序仍然在运行。要用 IDE 的方块型的停止按钮来真正的停止程序（或者在调试菜单里选择“停止调试”）。

6 让它停止

关闭程序时让你在第五步骤添加的循环停止。把外层循环修改为下面这样：

while (Visible)

现在运行程序并点击右上角的X按钮。
窗口关闭了，然后程序也停止了！
只是... IDE回到编辑模式之前有几秒钟延迟。

当你在if指令或循环里检查一个像
Visible一样的布尔值时，有时候就
相当于测试 (Visible == true)。
可以省略“==true” --只写布尔值就够了。

当你操作窗体或控件时，
只要窗体或控件还在被
显示Visible值就是真的。
如果你把它设置为假，
那将会使窗体或控件消失。

提示：&&运算符意味着
“AND”。你用它来把多个
条件测试连接成一个
大的条件测试，只有在
第一个为真AND第二个为
真AND第三个为真...时
它才为真。它很适合解
决这个问题。

你能弄明白什么导致了延迟
吗？你能修改它以使得程序
在窗口关闭时马上停止吗？



到时候用if/else语句做点练习了。 你能创建这个程序吗？

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Fun_with_If_Else
{
```

这是“Fun with if/else Statements!”

练习的整个Form.cs文件。我们要给你展示像这样的很多代码时，我们会在你需要添加的代码上画一个灰色框。

这儿是窗体的代码。我们把解决方案命名为“Fun with if else”，所以IDE把命名空间取名为Fun_with_if_else。如果你给解决方案一个不同的名字，他就会有一个不同命名空间。

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

你双击按钮时，IDE向窗体添加这个叫做button1_Click()的方法。每次按钮被点击这个方法都会运行。

外层if指令检查复选框是否被勾选了。检查！

```
private void button1_Click(object sender, EventArgs e)
{
    if (checkBox1.Checked == true)
    {
        if (label1.BackColor == Color.Red)
        {
            label1.BackColor = Color.Blue;
        }
        else
        {
            label1.BackColor = Color.Red;
        }
    }
    else
    {
        MessageBox.Show("The box is not checked");
    }
}
}
```

内层if指令检查标签的颜色。如果现在是红色，它运行一条变成蓝色的指令。

如果标签的背景颜色不是红色，这条指令就执行，来设置背景颜色为红色。

如果复选框没有勾选这个消息框窗口就会弹出。

你可以从 www.headfirstlabs.com/books/hfcssharp/ 下载所有练习的答案。



Exercise Solution

咱们来创建一点炫的东西!

有时候我们不会给你显示全部代码，只显示修改的那一点。这个闪烁的项目的所有逻辑都在 `button1_Click()` 方法里面。

IDE 添加这个方法时，会在大括号之前添加一个额外的换行。有时我们会把大括号放在代码的同一行来节省空间——但是 C# 并不在意空间，所以这样做完全合法。

```
private void button1_Click(object sender, EventArgs e)
{
    while (Visible) {
        for (int c = 0; c < 254 && Visible; c++) {
            this.BackColor = Color.FromArgb(c, 255 - c, c);
            Application.DoEvents();
            System.Threading.Thread.Sleep(3);
        }
        for (int c = 254; c >= 0 && Visible; c--) {
            this.BackColor = Color.FromArgb(c, 255 - c, c);
            Application.DoEvents();
            System.Threading.Thread.Sleep(3);
        }
    }
}
```

外循环只要窗体还是可见的就持续运行。窗体关闭了，Visible 就为 false 了，while 循环也就停止了。

第一个循环使得颜色以一个方向变换，第二个 for 循环反过来变换，所以看起来变化很平滑。

我们用 && 运算符解决了延迟的毛病，让每个 for 循环都检查 Visible 属性。这样 Visible 为 false，循环就停止了。

因为 for 循环需要在 while 循环检查 Visible 是否为真之前停止，所以产生了延迟。你可以通过向每个 for 循环的条件测试添加 **&&Visible == true** 来解决延迟。

你的代码和我们的有点不同吧？解决任何编程问题都不止有一个途径——比如你可能用 **while** 循环替代 **for** 循环。如果你的程序能运行，你的练习就成功了！

泳池难题

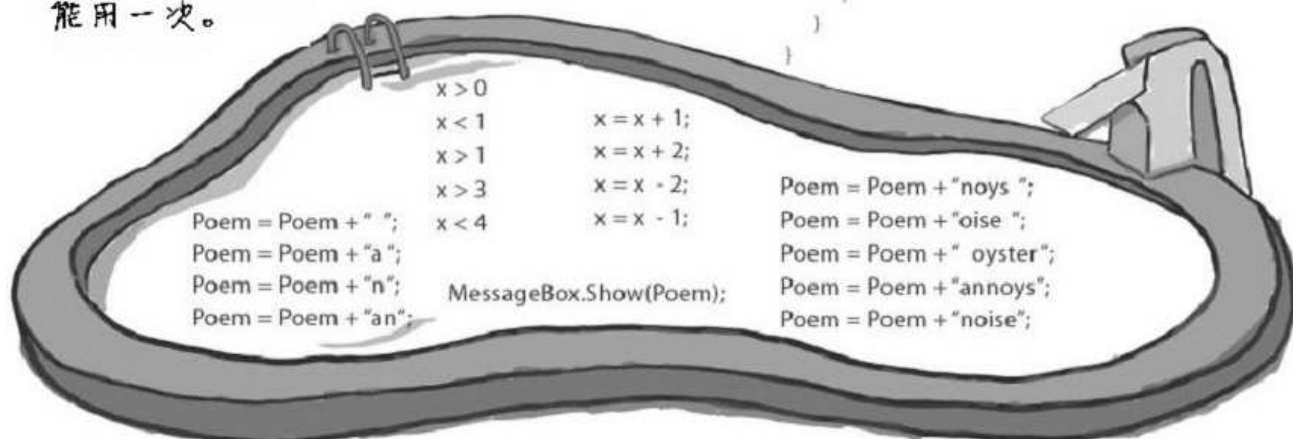


Output



我们在这本书里包含“泳池难题”来给你额外的训练。如果你喜欢小逻辑题，你会喜欢它的。如果你不是，也试一试吧——但是别担心，尽管看看答案吧。如果你在一个难题卡壳了，干脆跳过吧。

注意：每个只能
能用一次。



```
using System;
using System.Windows.Forms;
namespace Chapter_2 {
    class Chapter2PoolPuzzle {
        public static void Main() {
            int x = 0;
            String Poem = "";

            while ( _____ ) {
```

```
                if ( x < 1 ) {
```

```
                } _____
```

```
                if ( _____ ) {
```

```
                } _____
```

```
                if ( x == 1 ) {
```

```
                } _____
```

```
                if ( _____ ) {
```

```
                } _____
```

```
            } _____
```

```
        }
```

```
    }
```



代码贴士答案

一个C#程序的一部分粘在了冰箱上了。你能够把这些代码片重新排序形成一个可以运行显示一个消息框窗口的程序吗？有些括号掉在地上了，它们太小了捡不起来，所以尽管添加你认为需要的代码吧！

```
string Result = "";
```

```
int x = 3;
```

```
while (x > 0) {
```

```
{
```

```
if (x > 2) {  
    Result = Result + "a";  
}
```

```
x = x - 1;  
Result = Result + "-";
```

```
if (x == 2) {  
    Result = Result + "b c";  
}
```

这块儿没
掉在地上...

第一次进循环x
等于3，所以这
个条件测试为真。

第一次进入循环，这
条语句使得x等于2，
第二次进入循环使得
x等于1。

```
if (x == 1) {  
    Result = Result + "d";  
    x = x - 1;  
}
```

```
MessageBox.Show(Result);
```

Output:





泳池难题答案

```
using System;
using System.Windows.Forms;
namespace Chapter_2 {
    class Chapter2PoolPuzzle {
        public static void Main() {
            int x = 0;
            String Poem = "";

            while ( x < 4 ) {

                Poem = Poem + "a";
                if ( x < 1 ) {
                    Poem = Poem + " ";
                }
                Poem = Poem + "n";

                if ( x > 1 ) {

                    Poem = Poem + " oyster";

                    x = x + 2;
                }
                if ( x == 1 ) {

                    Poem = Poem + "noys ";
                }
                if ( x < 1 ) {

                    Poem = Poem + "oise ";
                }

                x = x + 1;
            }
            MessageBox.Show(Poem);
        }
    }
}
```

Head First C#中文版第一章

3对象：找对方向！

让代码意义明确

...这就是我的Husband类不需要HelpOutTheHouse()方法和PullHisOwnWeight()方法的原因。



你写的每一个程序都解决一个特定的问题。

写程序的时候，先想一想程序要解决什么问题是个好注意。这就是对象这么重要的原因。它让你可以基于要解决的问题来组织代码，这样你就可以集中精力于思考程序要解决的问题而不会被书写代码的技术细节纠缠住。正确的使用对象，你可以写出凭直觉的代码，它会易读易改。

Mike 是怎么思考他的问题的

Mike 是一个程序员，他正要去参加一个面试。他已经等不及要去秀一秀 C#技巧了，他是他得先去面试地点--并且他快要迟到了！

① Mike 想出了去面试的路径



我要走三十一号大桥，再去解放大街，然后走布卢姆菲尔德。

Mike 想好了目的地，然后找出了一条路径。

② 还好他带着收音机。交通大堵塞将会使他迟到了。

Mike 得到了新消息，他要避开一条大街。



这里是 Frank 为您播报天眼交通广播。看起来一起三车连环相撞事故已经使得解放大街到三十二大街彻底堵塞了。

③ Mike 想出了一条新路

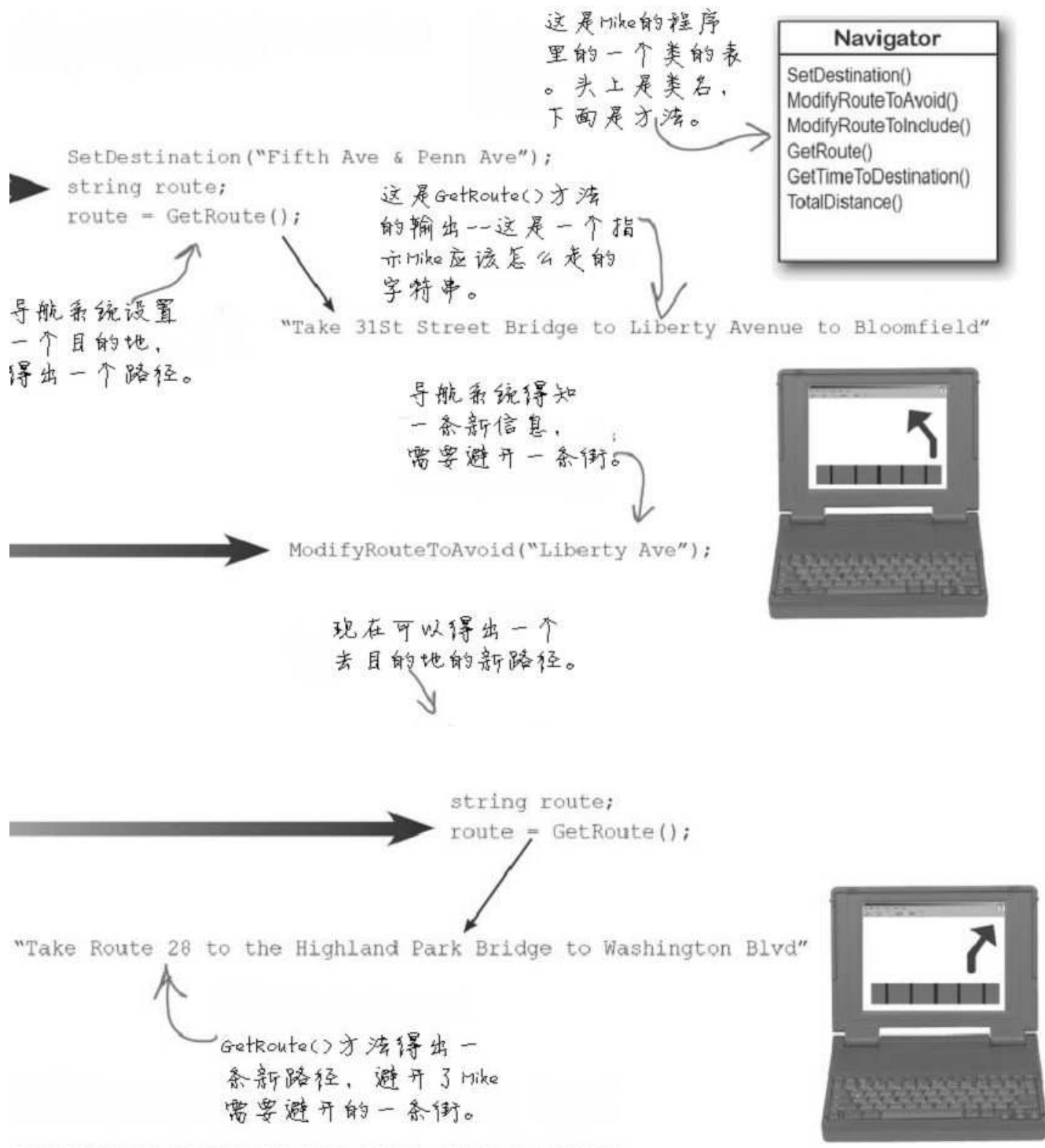
现在他想出来一条新路。

没问题，走老路一样不会迟到！



Mike 车上的导航系统怎么思考问题

Mike 自己编制了车辆导航系统，用来给自己在城里导航。



Mike 的街道导航系统和 Mike 以

相同的方法解决问题。

Mike 的导航类有用来设置和修改路径的方法

Mike 的导航器类有一些方法，功能就是在里面实现的。但是和你知道的 `button_Click()` 方法不一样，他们都围绕着一个问题：在城市中导航一条道路。这也就是 Mike 要把它们归为一类，并称该类为 `Navigator` 的原因了。

Mike 设计导航器类来方便的生成和修改路径。要得到一条路径，先要调用 `SetDestination()` 方法来设置目的地，然后用 `GetRoute()` 方法把路径信息包含到一个 `String` 中去。如果需要改变路径，他的程序调用 `ModifyRouteToAvoid()` 方法来避开一条特定的街道，然后再调用 `GetRoute()` 方法来得到新方向。

```
public class Navigator() {
    public void SetDestination(string destinationName) { ... };
    public void ModifyRouteToAvoid(string streetName) { ... };
    public string GetRoute() { ... };
}
```

Mike选择的方法名是意义明确的，让想要用它
在城里导航的人可以明白方法明的含义。

这个是方法的返回值。这意味着调用此方法的
语句可以用`GetRoute()`方法的返回值设置
一个`String`变量，让这个`String`包含路径信息
。如果返回值是`void`，这意味着方法没有返回值。

```
string route;
route = GetRoute();
```

有的方法有一个返回值

每个方法都是用语句组成的。有的方法只是执行完了语句就退出了。但是有的方法就有一个返回值，或者说一个在方法内部生成的或者计算得来的值，并把它送回到调用它的语句去。返回的值的类型（比如 `String`、`int`）叫做返回类型。

`return` 语句告诉方法马上退出。如果你的方法没有返回值--也就是说返回类型为 `void`--那么 `return` 语句就用分号结束，或者干脆不写 `return` 语句都可以。但是如果方法有返回类型，那么就必须有 `return` 语句。

这是个有返回值的
方法--返回一个
`int`。这个方法用
两个参数计算出结
果，用`return`语句
把值返回给调用它
的语句。

```
public int MultiplyTwoNumbers(int firstNumber, int secondNumber) {
    int result = firstNumber * secondNumber;
    return result;
}
```

这个语句调用方法来计算两个数字相乘。它返回一个 int:

```
int myResult = MultiplyTwoNumbers(3, 5);
```

方法可以接受3、5这样的值，也可以传参数给它。

用你所学的构建一个简单的应用

我们在一个类里面创建一个窗体，让它的按钮调用一个该类中的方法。



- 1 在 IDE 中创建一个新的窗体应用。然后添加一个叫做 Talker.cs 的类文件。你把类文件命名为 Talker.cs，IDE 就会自动把文件里面的类命名为 Talker。然后，IDE 就会在一个新标签页里面弹出这个类。
- 2 在类文件开头添加一句 using System.Windows.Forms; 然后向类里面添加代码：

```
class Talker {
    public static int BlahBlahBlah(string thingToSay, int numberOfTimes) {
        string finalString = "";
        for (int count = 1; count <= numberOfTimes; count++) {
            finalString = finalString + thingToSay + "\n";
        }
        MessageBox.Show(finalString);
        return finalString.Length;
    }
}
```

这一句声明一个叫做 final string 的变量，并设定它为空字符串。

BlahBlahBlah() 方法的返回值是它所输出的字符串的长度。你可以对任何字符串应用 ".Length" 来得出它的长度。

这个新类的 BlahBlahBlah() 方法接受两个参数。第一个参数告诉它说什么，第二个指示说几遍。它被调用的时候将会弹出一个消息框，带有重复的字符串信息。它的返回值是字符串的长度。它的 thingToSay 参数接受 String，numberOfTimes 参数接受一个数字。它将会在窗体的 TextBox 控件和 NumericUpDown 控件里得到这两个参数。

3 向你的项目里面添加这个窗体

给这个TextBox的默认值设置为“Hello”



然后双击按钮让它含有这些代码:

```
int len = Talker.BlahBlahBlah(textBox1.Text, (int) numericUpDown1.Value);
MessageBox.Show("The message length is " + len);
```

4 现在运行程序! 点击按钮, 你会看到弹出两个消息框。类本身弹出第一个, 第二个是窗体弹出的。

BlahBlahBlah ()
方法根据两个参数
弹出这个消息框。



方法返回值的时候, 窗体就会弹出这个消息框。



把这个
NumericUpDown
控件的最小值设为1, 最大值设为10; 默认值3.

you are here >

89

Mike 有个好主意

面试进行的很顺利! 但是早上的交通堵塞让 Mike 想要改进他的导航器。



Mike 可以创建三个不一样的导航器类...

Mike 可以把 Navigator 类的代码复制进另外两个类里面。这样他的程序就可以同时存储三条路径了。

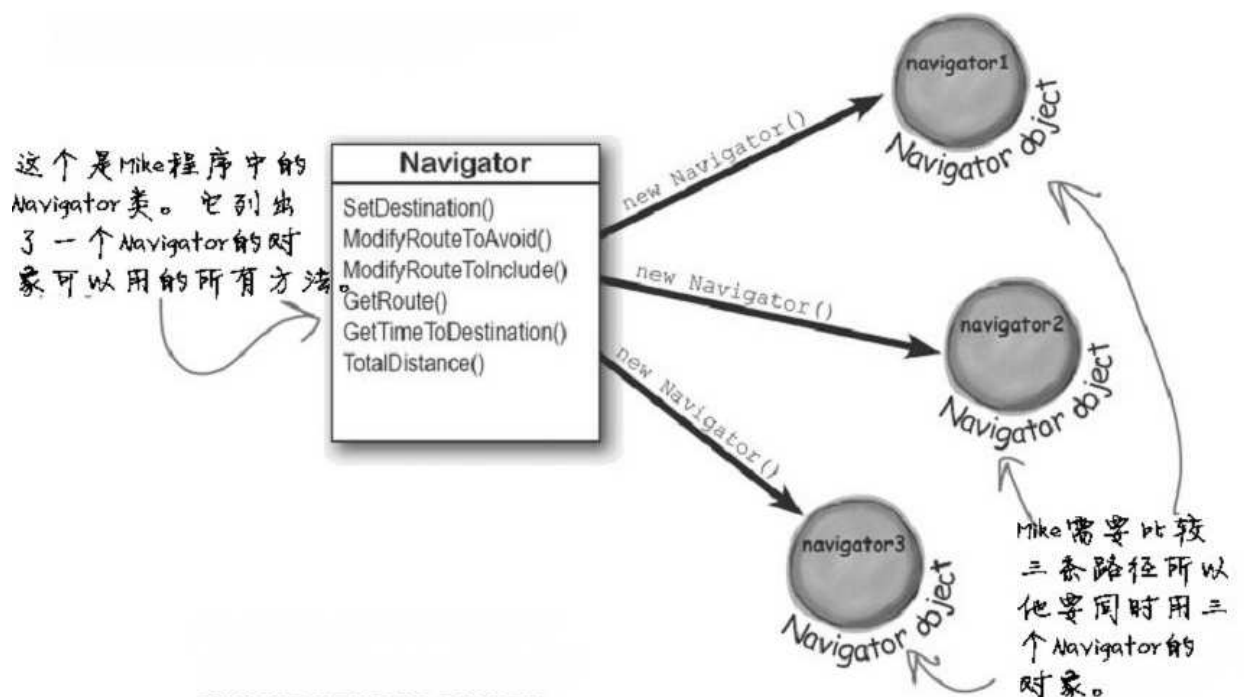


对！同时维护三份一样的代码太麻烦了。

很多你需要解决的问题都需要反复运用一件事物。在这儿是很多条路径。但是也有可能会是涡轮、狗、音乐文件，任何事物。这些程序有一个共同点：用同样的方式对待同样的事物，不管这样的事物有多少个。

Mike 可以用对象解决他的问题

在 C# 中你可以运用对象作为处理相似事物的工具。Mike 可以只编写一次，但是多次的使用，随你需要。



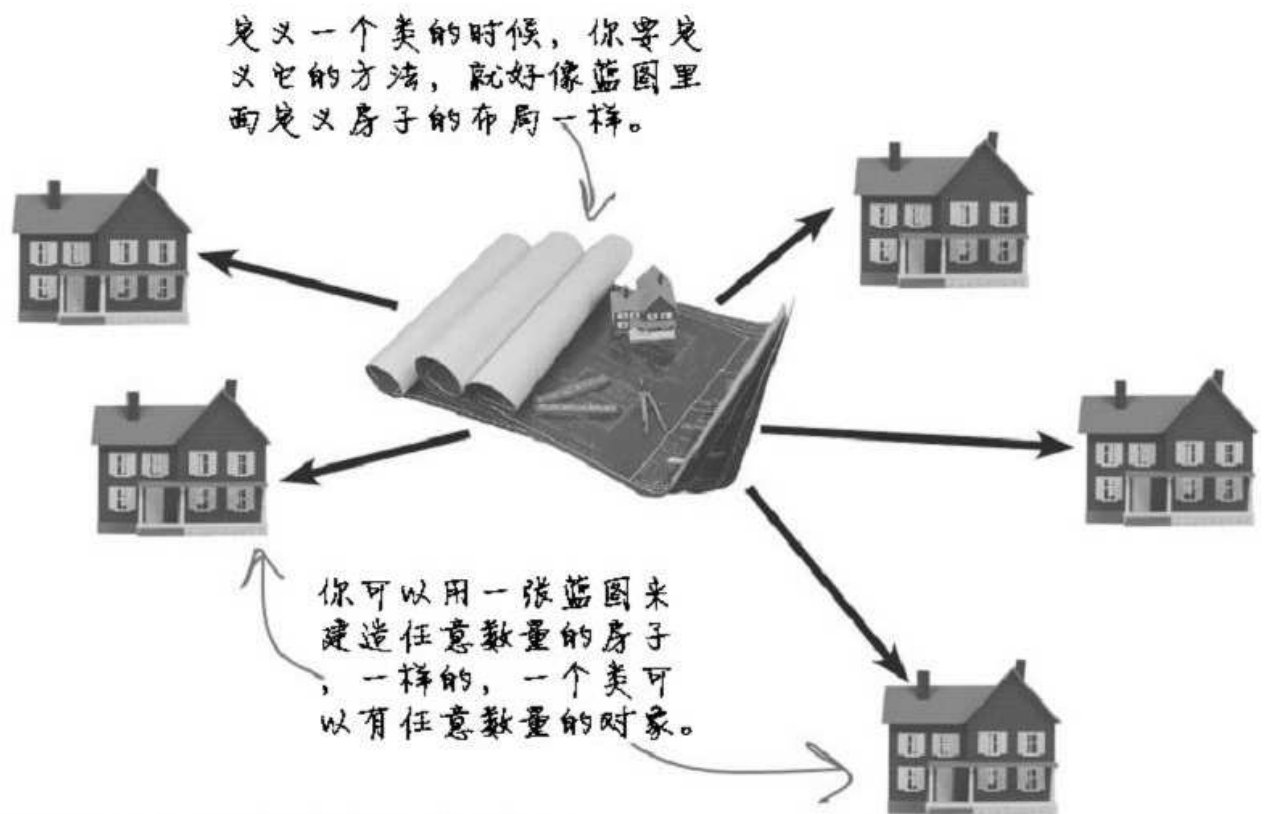
创建一个对象只需要用 **new** 关键字和一个类的名字。

```
Navigator navigator1 = new Navigator();
navigator1.SetDestination("Fifth Ave & Penn Ave");
string route;
route = navigator1.GetRoute();
```

现在你可以用这个对象了！你创建了一个类的对象，这个对象就可以用类里面定义的所有方法。

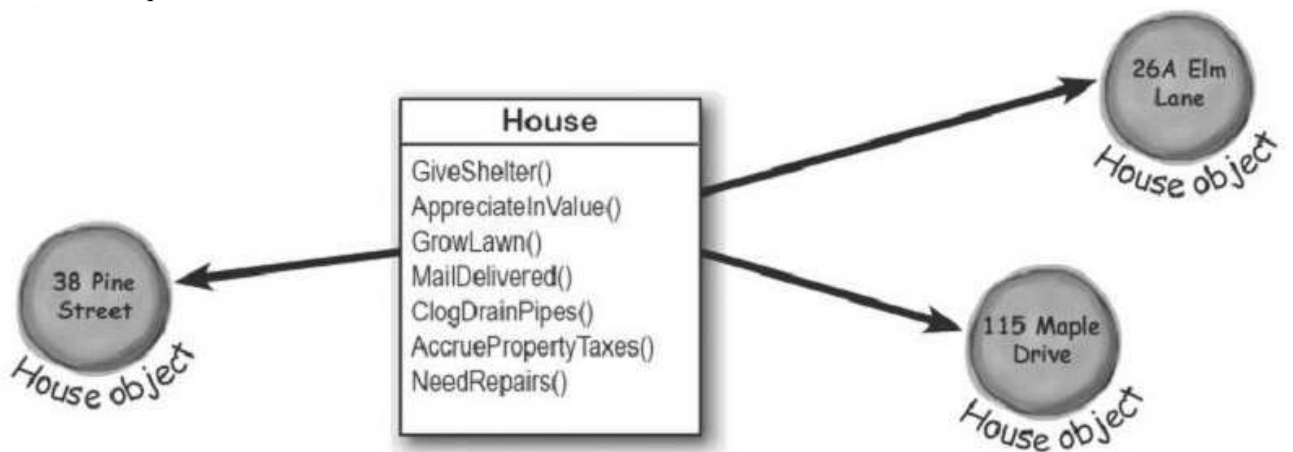
你是用类来生成对象

类就像是对象的蓝图。如果要在一个市郊房地产开发中建五座一样的房子，你不会请建筑师来画五个一样的蓝图。只要用一张蓝图建五座一样的房子就可以了。



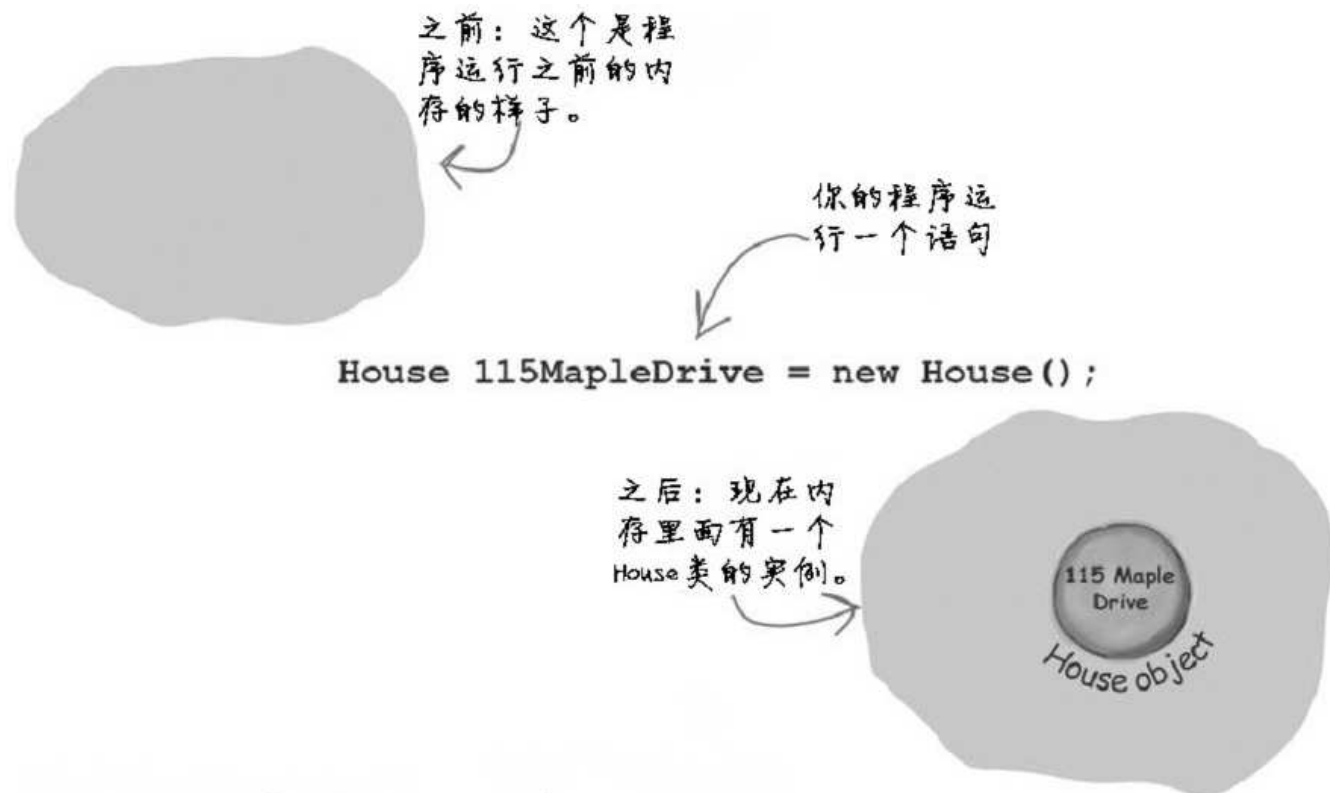
一个对象是从它的类里面得到的方法

一旦写好一个类，你可以用 `new` 语句来创建任意数量的对象。这样创建了对象之后，类里面的每一个 `public` 方法都是对象的一部分。



一个类的对象也叫做那个类的实例

你猜怎么样...你已经见过了！工具箱里的每个东西都是类：有 `Button` 类、`TextBox` 类、`Label` 类，等等。你从工具箱里面拖拽出来一个 `Button`，IDE 就会自动创建一个它的实例并命名为 `button1`。在拖拽一个 `Button`，将会在创建一个实例叫做 `button2`。每一个 `Button` 的实例都有自己的属性和方法。但是每一个按钮行为都是一致的，因为他们都是同一个类的实例。



自己查看一下！

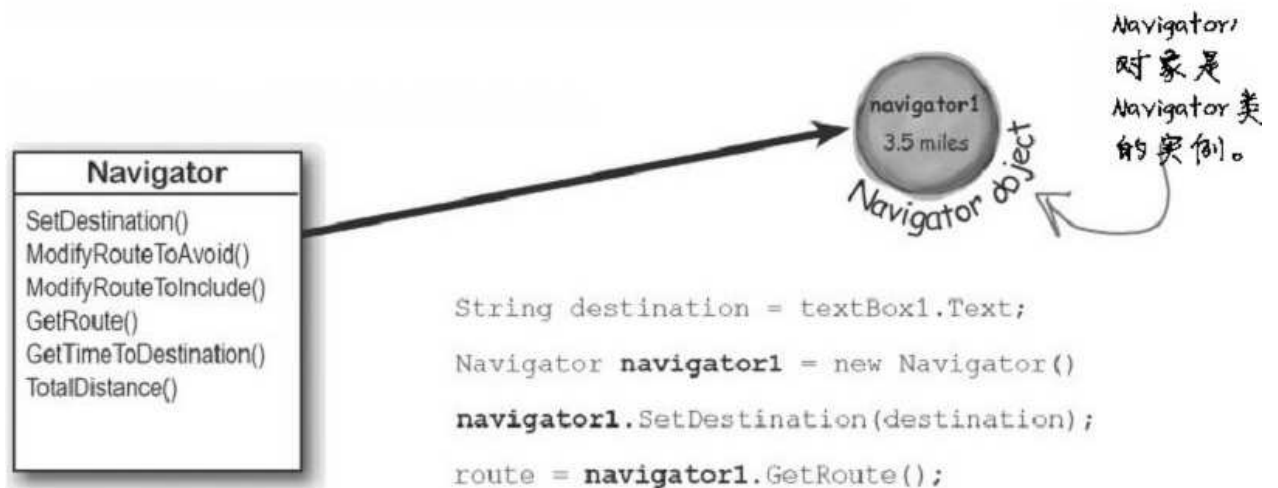
打开一个含有叫做 button1 的按钮实例的工程，在 IDE 里面搜索“new Button”。你将在窗体设计器里发现创建 Button 类的实例的代码。

由对象带给你的...更好的解决方案！

Mike 想出了一个新的路径比较程序，它用对象来找出三条目的一样的路径中最短的一条。下面是它构建程序的方式。

GUI的意思是图形用户接口，你在窗体设计器里面做窗体的时候就是在做GUI。

- 1 Mike 设计的 GUI 中有 TextBox--textBox1 中填写目的地。然后添加 TextBox2，填写需要避开的街道；TextBox3 填写第三条路径必须经过的街道。
- 2 他创建一个 Navigator 对象并设置目的地。



3 然后创建第二个 Navigator 对象，叫做 Navigator2.调用 SetDestination () 方法来设置目的地，然后调用 ModifyRouteToAvoid () 方法。

SetDestination
, ModifyRouteToAvoid
和 ModifyRouteToInclude
方法都接受一个 string
作为参数。

4 第三个 Navigator 对象叫做 Navigator3.Mike 设置它的目的地，然后调用它的 ModifyRouteToInclude () 方法。



5 现在 Mike 可以调用每一个对象的 TotalDistance () 方法来看看到底哪一个路径最短。他只需要写一次代码，而不是三次！

创建一个类的对

象就叫做创建这

个类的一个实例。



按照 Mike 做的步骤来写代码，创建 Navigator 对象并调用它们的方法。

```
String destination = textBox1.Text;
String route2StreetToAvoid = textBox2.Text;
String route3StreetToInclude = textBox3.Text;

Navigator navigator1 = new Navigator()
navigator1.SetDestination(destination);
int distance1 = navigator1.TotalDistance();
```

我们帮你开一个头儿。
Mike 用这些代码来在
文本框里获得目的地
和街道名字。

这儿的代码创建 Navigator
对象，设置它的目的地，获
得距离。

1. 创建 Navigator2 对象，设置它的目的地，调用
ModifyRouteToAvoid() 方法，并用它的 TotalDistance() 方法来
给一个叫做 distance2 的 int 变量赋值。

```
Navigator navigator2 = .....

navigator2. ....

navigator2. ....

int distance2 = .....
```

2. 创建 Navigator3 对象，设置它的目的地，调用
ModifyRouteToInclude() 方法，并用它的 TotalDistance() 方法
来给一个叫做 distance3 的 int 变量赋值。

```
.....

.....

.....

.....

.....
```

C# 内建的 Math.Min() 方法比较两个数字，并返回较小的值。Mike 用
它来找出哪条路最短。

```
int shortestDistance = Math.Min(distance1, Math.Min(distance2, distance3));
```

Sharpen your pencil 答案

按照 Mike 做的步骤来写代码，创建 Navigator 对象并调用它们的方法。

```
String destination = textBox1.Text;
String route2StreetToAvoid = textBox2.Text;
String route3StreetToInclude = textBox3.Text;

Navigator navigator1 = new Navigator()
navigator1.SetDestination(destination);
int distance1 = navigator1.TotalDistance();
```

我们给了你一个开头。这儿的代码得到目的地和街道名字，这段代码还创建 Navigator 对象，设置它的路径，得到距离。

1. 创建 Navigator2 对象，设置它的目的地，调用 ModifyRouteToAvoid() 方法，并用它的 TotalDistance() 方法来给一个叫做 distance2 的 int 变量赋值。

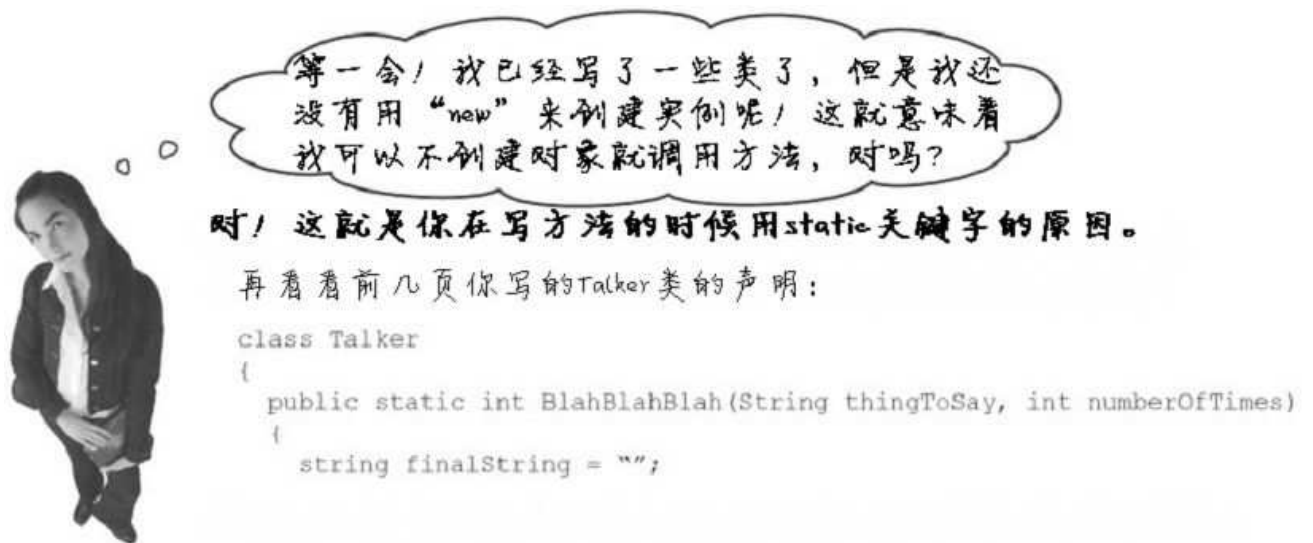
```
Navigator navigator2 = new Navigator()
navigator2.SetDestination(destination);
navigator2.ModifyRouteToAvoid(route2StreetToAvoid);
int distance2 = navigator2.TotalDistance();
```

2. 创建 Navigator3 对象，设置它的目的地，调用 ModifyRouteToInclude() 方法，并用它的 TotalDistance() 方法来给一个叫做 distance3 的 int 变量赋值。

```
Navigator navigator3 = new Navigator()
navigator3.SetDestination(destination);
navigator3.ModifyRouteToInclude(route3StreetToInclude);
int distance3 = navigator3.TotalDistance();
```

C# 内建的 Math.Min() 方法比较两个数字，并返回较小的值。Mike 用它来找出哪条路最短。

```
int shortestDistance = Math.Min(distance1, Math.Min(distance2, distance3));
```



你不用创建 Talker 类的实例就可以调用方法。只需要这样：

```
Talker.BlahBlahBlah("Hello hello hello", 5);
```

Static 方法就是这么调用的，你已经这么做过。如果你把 BlahBlahBlah() 方法的 static 关键字去掉，那你就一定要创建一个 Talker 的实例来调用这个方法。除了这个区别，静态方法和实例方法一样。你可以给它传递参数，它会返回值，并且它处在类里面。

你还可以用 static 关键字做一件另外的事儿。你可以把整个类声明为 static，那么它的所有方法一定要是静态的。如果给一个静态类添加一个非静态方法，会无法编译的。

there are no Dumb Questions

问：我想到静态就想到了不会改变的东西。这意味着非静态方法可以改变而静态方法不能改变吗？它们的行为方法有区别？

答：不，静态的和非静态的方法行为方式是一样的。唯一区别是静态方法不要求一个实例，而非静态方法要有实例才行。很多人记不住这一点，因为“静态”这个词汇不是很直观。

问：如果我不创建我的类的一个实例我就不能用它？

答：可以用它的静态方法。但是非静态方法就需要创建实例了。

问：那我干嘛要写需要实例的方法呢？我为什么不把所有的方法都写成静态的？

答：因为如果你的对象需要保持一些数据--比如 Mike 的每个 Navigator 对象都需要保持一个路径的数据--这样你才可以用每一个对象来操作不同的数据。所以当 Mike 调用 Navigator2 对象的 ModifyRouteToAvoid() 方法的时候，只会影响保存在这这个实例里面的路径数据，不会影响到 Navigator2 和 Navigator3 对象。这样 Mike 就可以同时操作三条路径--它的程序就可以同时跟踪它们。

问：实例是怎么保持数据的？

答：翻过页来找答案！

实例用字段来保持数据

你在 IDE 中通过设置按钮的 Text 属性来改变按钮上的文本。你这样做的时候，IDE 添加这

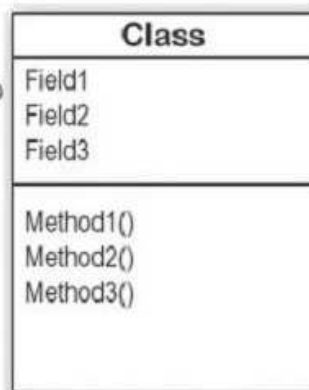
样的代码：

```
button1.Text = "Text for the button";
```

从技术上来讲，这是在设置属性。属性是一种特殊的字段--但是我们一会儿在详细

现在你知道 button1 是 Button 类的实例。上面的代码就是在修改 button1 实例的字段。你可以向类图添加字段--只需要在类图中间划一道水平线，线上是字段，下面是方法。

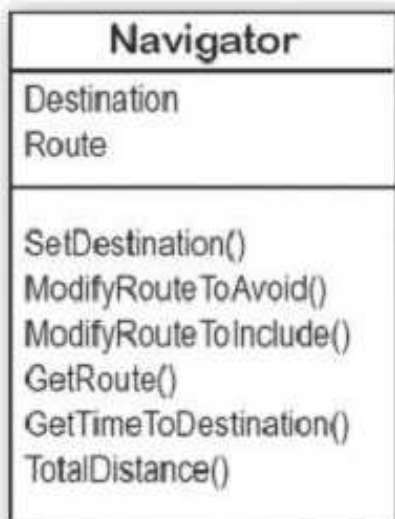
类图在这个位置表示自己的字段。这个类的每一个实例用它们来保持自己的状态。



添加这条线来分割字段和方法。

对象可以做的是方法，知道的是字段。

Mike 创建了 Navigator 类的三个事例，他的程序创建三个对象。每个对象用来保持一个不同的路径。程序创建 Navigator2 实例并调用它的 SetDestination（）方法的时候，这只是设置了这一个实例的目的地。但是不会影响到 Navigator1 和 Navigator3 实例。



每一个 Navigator 实例都知道自己的目的地和路径。

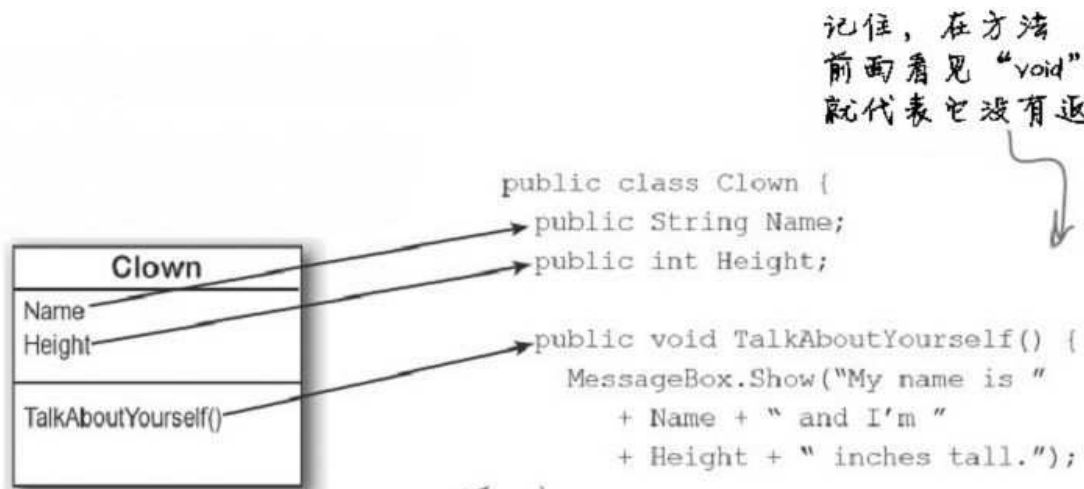
Navigator 对象可以做的事情是让你设置目的地，修改路径，并得到关于路径的信息。

一个对象的行为是用它的方法定义

的，并且它用字段来保持自己的状态。

我们来创建一些实例！

向类里面添加字段很容易。只要在方法外面声明变量就可以了。这样每一个实例都可以得到这些变量的一份拷贝。



记住，在方法前面看见“void”就代表它没有返回值。

如果你想要让你的类可以创建实例，那就不要给类或者方法添加static关键字。

记住，*=运算符取其左侧的值，并给它乘以右侧的值。



Sharpen your pencil

写出在MessageBox中将会输出的内容

```

Clown oneClown = new Clown();
oneClown.Name = "Boffo";
oneClown.Height = 14;

```

```
oneClown.TalkAboutYourself();
```

"My name is _____ and I'm _____ inches tall."

```

Clown anotherClown = new Clown();
anotherClown.Name = "Biff";
anotherClown.Height = 16;

```

```
anotherClown.TalkAboutYourself();
```

"My name is _____ and I'm _____ inches tall."

```

Clown clown3 = new Clown();
clown3.Name = anotherClown.Name;
clown3.Height = oneClown.Height - 3;

```

```
clown3.TalkAboutYourself();
```

"My name is _____ and I'm _____ inches tall."

```
anotherClown.Height *= 2;
```

```
anotherClown.TalkAboutYourself();
```


"My name is _____ and I'm _____ inches tall."

感谢内存

你的程序创建一个对象之后，对象就处于电脑的一个区域中，叫做堆。你的程序用 `new` 语句创建一个对象，C#就会在堆中为它分配一块空间来存储对象的数据。

这是对象被创建之前的堆的样子。注意，它是空的。

咱们来近距离看看怎么回事儿

**Sharpen your pencil**
Solution

写出会输出的内容

```
Clown oneClown = new Clown();
oneClown.Name = "Boffo";
oneClown.Height = 14;
oneClown.TalkAboutYourself();

Clown anotherClown = new Clown();
anotherClown.Name = "Biff";
anotherClown.Height = 16;
anotherClown.TalkAboutYourself();

Clown clown3 = new Clown();
clown3.Name = anotherClown.Name;
clown3.Height = oneClown.Height - 3;
clown3.TalkAboutYourself();

anotherClown.Height *= 2;
anotherClown.TalkAboutYourself();
```

每一个 `new` 语句创建一个 `Clown` 的实例，并在堆里面为它预分配一块空间并用对象的数据填充这块空间

"My name is Boffo and I'm 14 inches tall."

"My name is Biff and I'm 16 inches tall."

"My name is Biff and I'm 11 inches tall."

"My name is Biff and I'm 32 inches tall."

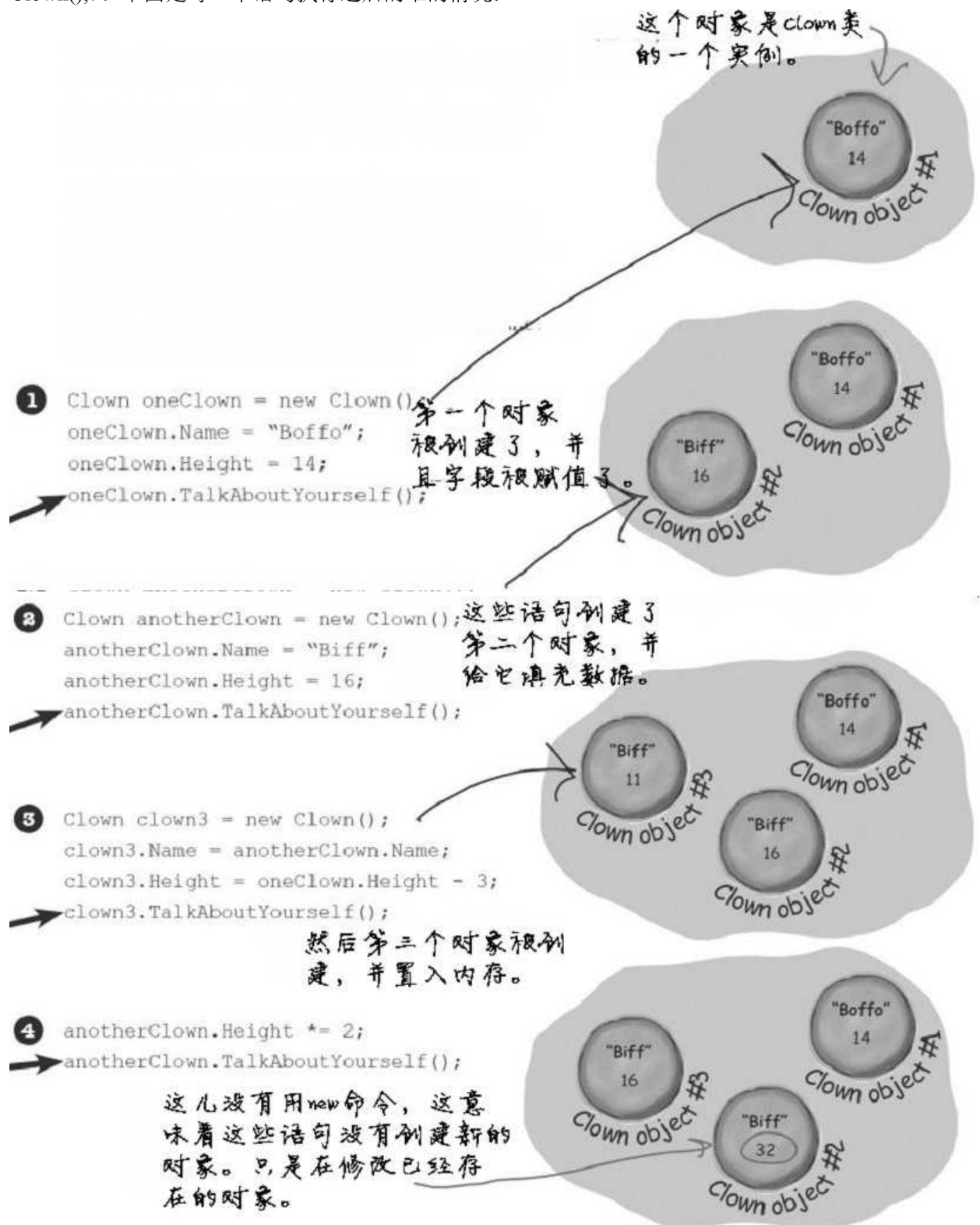
你的程序创建的对象会被添加到堆

你的程序心里在想什么

你的程序就是这样创建一个 `Clown` 类的实例的：

```
Clown myInstance = new Clown();
```

实际上上面是两个语句连在了一起。第一个语句声明一个 Clown 类型的变量 (Clown myInstance;)。第二个语句创建一个对象并把它赋值给刚刚创建的变量 (myInstance = new Clown();)。下面是每一个语句执行之后的堆的情况:



你可以给类和方法命名来让代码直观

把代码放在方法内，就是在选择到了程序的结构。你用过方法吗？你会把一个方法分裂到多个方法吗？你需要方法吗？你对方法做的选择会让你的代码更直观--如果你不细心的话，反而就更费解了。

1 这儿是一段紧凑的代码。这段代码来自于一个控制生产糖果的程序。

“obj”, “ics”, 和
“m”是差劲的名
字！我们不知道
它们是做什么的
。还有，T()
类是干嘛的？

```
int t = m.chkTemp();
if (t > 160) {
    T obj = new T();
    tb.clsTrpV(2);
    ics.Fill();
    ics.Vent();
    m.airsyschk();
}
```

chkTemp()方法返回
一个整数...但是它
内部做什么？

clsTrpV()方法接
受一个参数，但
是我们也不知道
这个参数什么含义。

再看一会儿。你知道这段代码是做什么的吗？

2 从这段代码你看不出来它是做什么的。在这个情况下，程序员很乐于把所有代码都写到一个方法。但是把代码弄得尽可能紧凑并不是很有用！我们来把它分到多个方法来让它易读，缤纷确定把类名命名的有意义。我们还是从分析代码做什么来开始。

你怎么分析代码是干
嘛的？写代码都是有
原因的。分析原因是
你的任务！在这个例
子里，我们可以通过
阅读规格说明来分析
代码。

通用5型电子糖果制造机
规格手册

奶油杏仁糖的温度需要每三分钟自动检查
一次。如果温度超过160度，糖果就过热
了。此时需要启动冷却通风。

- * 关闭二号节流阀
- * 给冷却系统充水
- * 排水
- * 确定没有残留空气

3 那一页手册使得代码容易理解了。这提示了我们怎么把我们的代码写的更易懂。现在我们知道为什么代码里有一个比较 t 和 160 的条件测试--手册说温度高于 160 对于糖果来说过热了。也看明白了“m”是一个控制糖果机的类，它有测试糖果温度的静态方法。我们来把温度检测放进一个方法，并给类和方法命名来让它们的名字有意义。

```

public boolean IsNougatTooHot() {
    int temp = Maker.CheckNougatTemperature();
    if (temp > 160) {
        return true;
    } else {
        return false;
    }
}

```

IsNougatTooHot () 方法的返回值。

给类命名为Maker，给方法命名为CheckNougatTemperature这样代码更易读。

方法的返回值是boolean，这意味着它返回一个true或者false值。

4 如果糖果过热了，手册指示说怎么办？它告诉我们启动冷却系统。我们再写一个方法，并给 T 类和 ics 类取一个好名字。

```

public void DoCICSVentProcedure() {
    Turbine turbineController = new Turbine();
    turbineController.CloseTripValve(2);
    IsolationCoolingSystem.Fill();
    IsolationCoolingSystem.Vent();
    Maker.CheckAirSystem();
}

```

void返回值意味着方法不返回任何值。

5 现在代码直观多了！即使你不了解冷却系统，这段代码也很容易理解了：

```

if (IsNougatTooHot() == true) {
    DoCICSVentProcedure();
}

```

仔细考虑你的代码需要解决的问题可以使得你的代码易读、易写。如果你的方法名字对于了解这个问题的人有意义，那么你的代码将会更易懂...也易于开发！

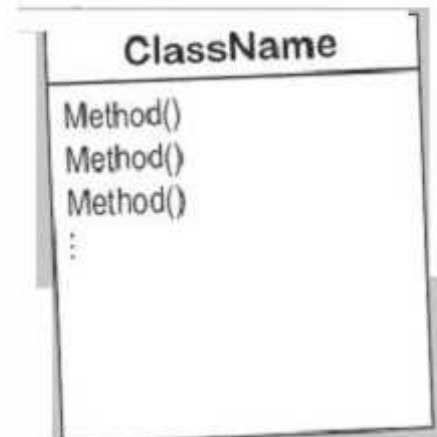
给你的类一个自然的结构

花一会时间想一想为什么你想要让你的方法直观：因为每个程序都解决一个问题或者有一个目的。有可能不是业务问题--有时候一个程序的目的只是耍耍酷或者为了好玩！无论你的程序要解决什么问题，你的代码写的越像你要解决的问题，你的程序就越容易写（容易读，容易修改，容易维护）。

用类图来设计你的类

类图是把类画在纸上的一个简单方法。它在你写代码值之前的设计阶段是一个有价值的工具。

在类图的最上部写类名。然后把方法名写在底部。现在你可以一目全览这个类了。



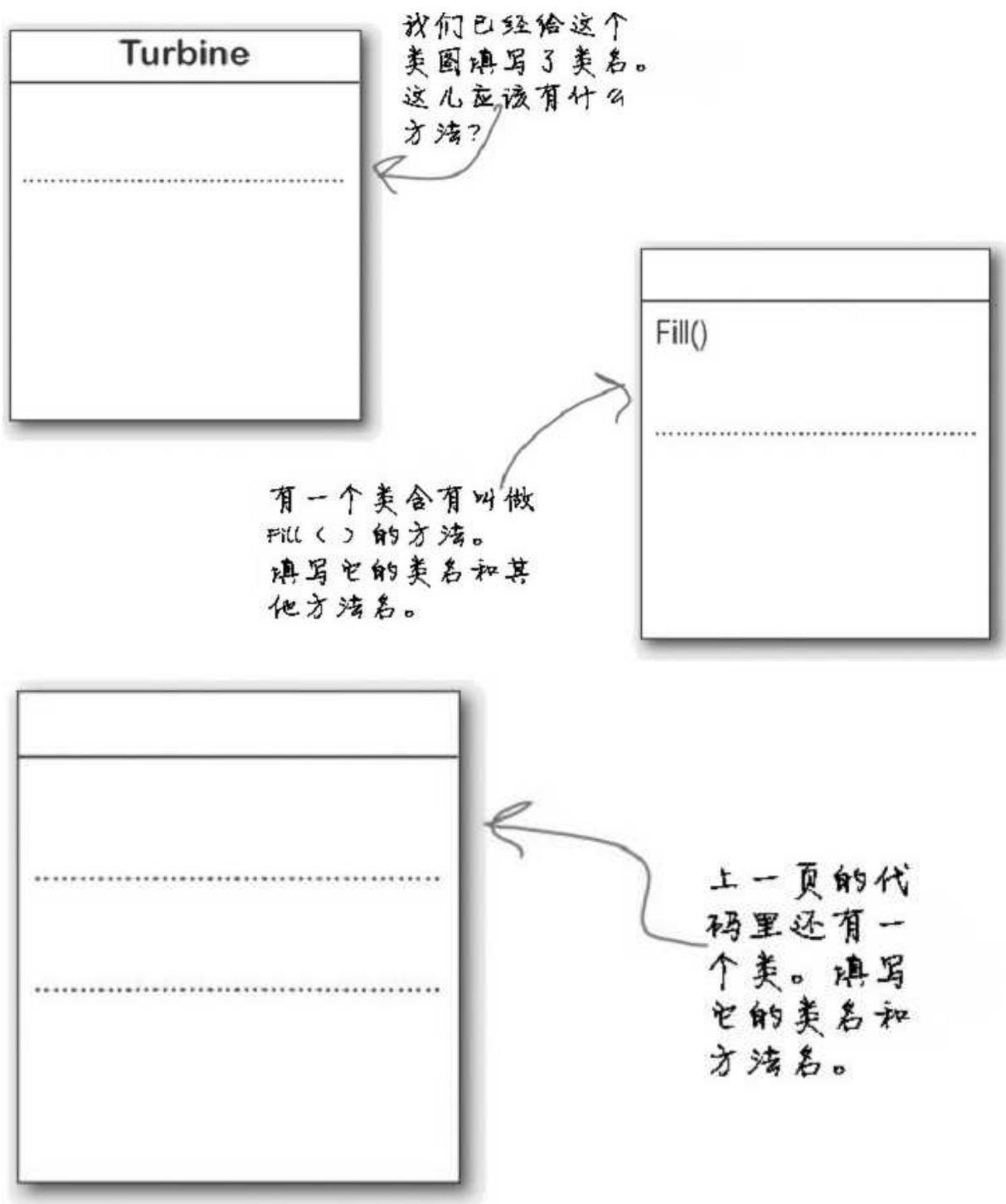
我们来创建一个类图

再看一下上一页的第五个步骤里的 if 语句。你已经知道语句都要在方法体里面，对吧？在这个例子里，if 语句处于 DoMaintenanceTests() 方法里面，而这个方法属于 CandyController 类。现在看看代码再看看类图。看出来它们之间的联系了么？

```
public class CandyController {  
    public void DoMaintenanceTests() {  
        ...  
        if (IsNougatTooHot() == true) {  
            DoCICSVentProcedure();  
        }  
        ...  
    }  
    public void DoCICSVentProcedure() ...  
    public boolean IsNougatTooHot() ...  
}
```

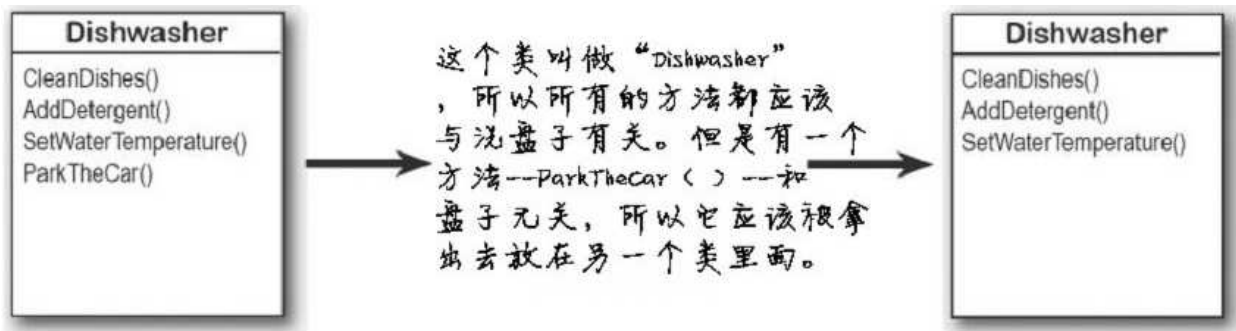


我们上一页写的糖果制造控制系统的代码调用另外三个类。向回翻页看看代码，并填写下面的类图。

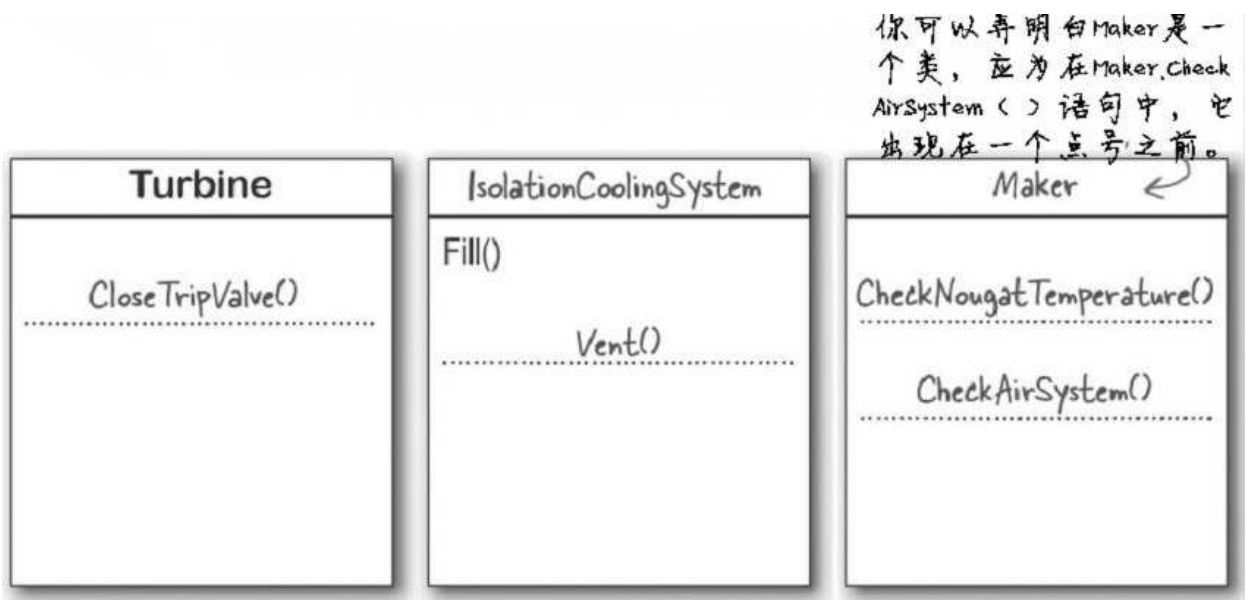


类图帮你把类组织的有意义

画类图帮你更容易的在写代码之前发现潜在的问题。在处理代码细节之前从一个更高的角度考虑可以帮你把代码结构组织的更集中于它要解决的问题。它会帮你避免写不必要的或者结构差劲的类和方法，而写出来的都是直观的和易用的。



我们上一页写的糖果制造控制系统的代码调用另外三个类。向回翻页看看代码，并填写下面的类图。



这些类都有一个严重的设计缺陷。写下你认为每个类有什么不对，并写下你打算怎么修改。

Class23
CandyBarWeight() PrintWrapper() GenerateReport() Go()

这个类是我们早先看见的糖果制造系统的一部分。

DeliveryGuy
AddAPizza() PizzaDelivered() TotalCash() ReturnTime()

DeliveryGirl
AddAPizza() PizzaDelivered() TotalCash() ReturnTime()

这两个类是一个比萨店系统用来送外卖的。

CashRegister
MakeSale() NoSale() PumpGas() Refund() TotalCashInRegister() GetTransactionList() AddCash() RemoveCash()

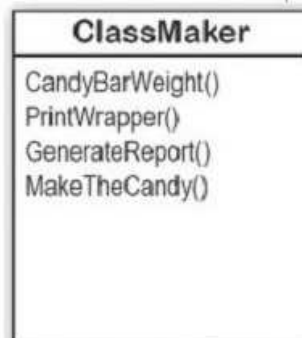
这个类是一个自动化便利店收费系统的一部分。



我们就是这样改正这些类的。这只是改正的一种方法--还有很多其他的方法来根据怎么用这些类来设计类。

这个类是我们先前见过的糖果制造机的一部分。

类名没有很好的形容类的作用。一个程序员如果看见 `Class23.Go()` 这一行，肯定看不出来这一行做什么。我们也需要给方法重命名为更有描述性的名字——我们可以给它命名为 `MakeTheCandy()`，不过你也可以给它命名为任何名字。



这两个类是一个比萨店的外卖程序的一部分。

看起来 `DeliveryGuy` 类和 `DeliveryGirl` 类都做一样的事情——他们都是送外卖的。一个更好的设计是把两个类归为一个类并添加一个说明性别的字段。



我们添加性别字段因为我们估计最开始有两个类的原因男女需要区别对待。

这个类是一个自动化便利店的收银台程序的一部分。

这个类的每一个方法做的事儿都和现金收取有关——卖东西啊，列出交易清单啊，添加现金啊... 只有一个例外：加油。把这个方法从这个类拖出去放进别的类是个好主意。



```
public partial class Form1 : Form
{
    private void button1_Click(object sender, EventArgs e)
    {
        String result = "";
        Echo e1 = new Echo();

        _____

        int x = 0;
        while ( _____ ) {
            result = result + e1.hello() + "\n";

            _____

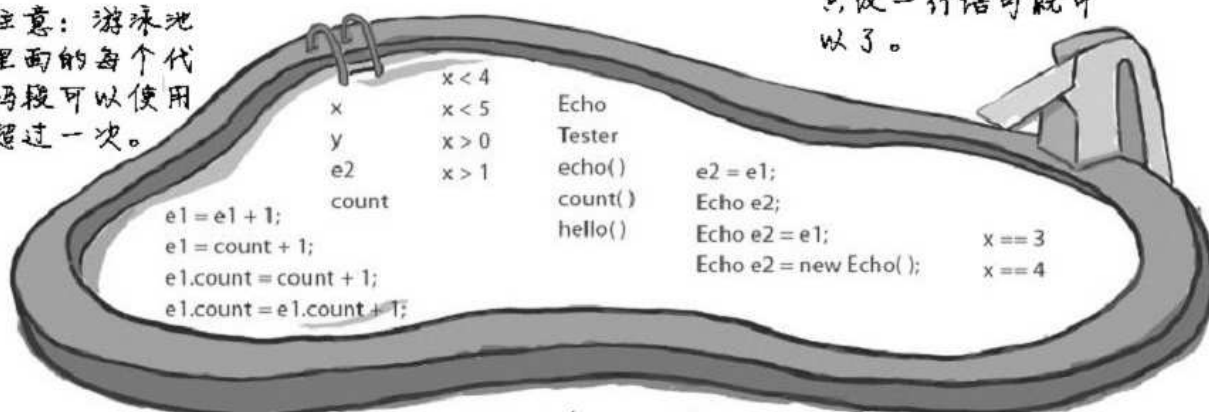
            if ( _____ ) {
                e2.count = e2.count + 1;
            }

            if ( _____ ) {
                e2.count = e2.count + e1.count;
            }

            x = x + 1;
        }
        MessageBox.Show(result + "Count: " + e2.count);
    }

    public class _____ {
        public int _____ = 0;
        public string _____ {
            return "helloooo...";
        }
    }
}
```

注意：游泳池里面的每个代码段可以使用超过一次。



→ 答案在 120 页

Pool Puzzle



你的任务是吧游泳池里面的代码片段取出来放进这儿的代码的空行里。同一个代码段可以用多次，但是没必要把所有的代码段都用到。你的目的是让类可以编译并运行并产生列出来的输出结果。

Output



额外的问题！

如果输出的最后一行是24而不是10，应该怎么解题呢？只改一行语句就可以了。

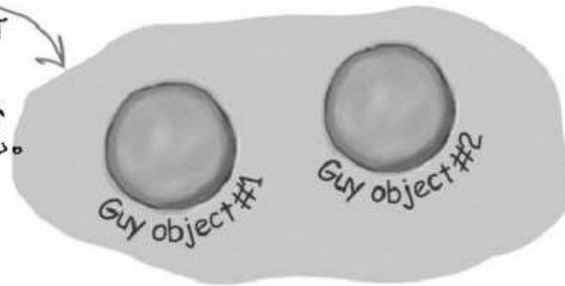
创建一个类来和一些共事

Joe 和 Bob 经常互相借钱。我们来创建一个类来与他们共事。

1 我们创建一个 **Guy** 类并创建它的两个实例放到一个窗体里面

窗体里有两个字段，一个叫做 joe（代表第一个人），另一个叫做 bob（代表第二个人）。

创建这两个对象的new语句，在窗体初始化的时候就得以运行。这儿是窗体载入之后的堆的情况。

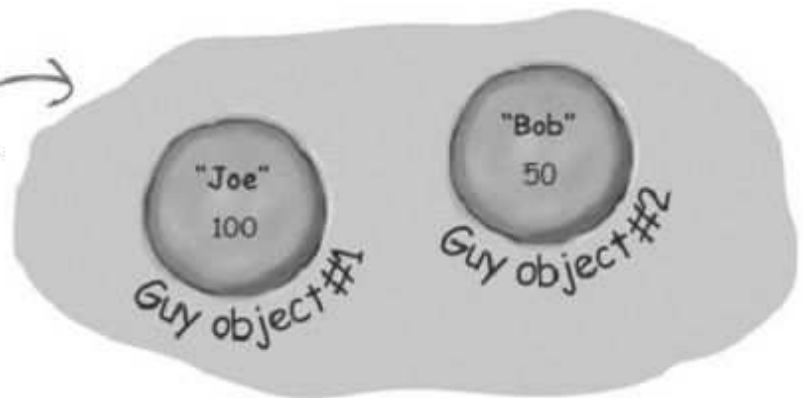


Guy
Name
Cash
GiveCash()
TakeCash()

2 我们将会设置 Guy 对象的现金和名字

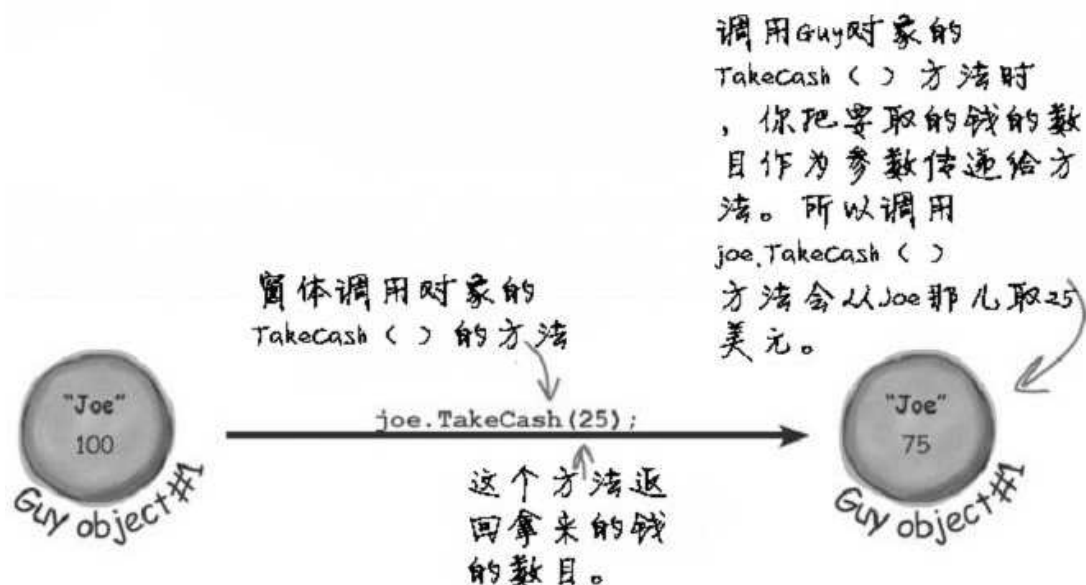
这两个对象代表不同的人，所以每个对象有不同的名字和不同的现金。

每个人有一个Name字段保持他的名字，还有一个cash字段保持他的现金数目。



3 我们将会给他们现金还会从他们那儿拿钱出来

我们将会用每个人的 GiveCash（）方法来给每个人现金，还会用 TakeCash（）方法来从他们那儿拿钱。



创建你的 Guy 类的一个对象

创建一个新的窗体应用（因为我们要用到一个窗体）。然后用解决方案管理器添加一个叫做Guy 的类。确定要做 Guy 类文件里添加“using system.Windows.Form;”，然后写完这个类。下面是代码：

```
public class Guy {
    public string Name;
    public int Cash;

    public int GiveCash(int amount) {
        if (amount <= Cash && amount > 0) {
            Cash -= amount;
            return amount;
        } else {
            MessageBox.Show(
                "I don't have enough cash to give you " + amount,
                Name + " says...");
            return 0;
        }
    }
}
```

Guy类有两个字段。Name字段是个string，它代表名字（“Joe”）。Cash字段是一个int，代表人有多少钱。

GiveCash()方法有一个参数，叫做amount，你用它指示给Guy的实例多少钱。

他用一个if语句检查是否有足够的现金——如果有，他就从自己的现金中取出钱，并把钱数作为返回值返回。

Guy要确定你想他要的钱数是否大于0，否则它将会增加自己的钱数而不是减少。

如果这个人没有足够的钱，它将会用消息框告诉你，然后返回一个0。

```

public int ReceiveCash(int amount) {
    if (amount > 0) {
        Cash += amount;
        return amount;
    } else {
        MessageBox.Show(amount + " isn't an amount I'll take",
            Name + " says...");
        return 0;
    }
}

```

ReceiveCash() 方法和 GiveCash() 方法工作方式类似。它有一个参数，确保参数大于0，然后把它添加到自己的现金。

如果 amount 是正数，ReceiveCash() 方法就返回添加的 amount 值。如果 amount 是 0 或负数，就会显示一个消息框并返回 0。

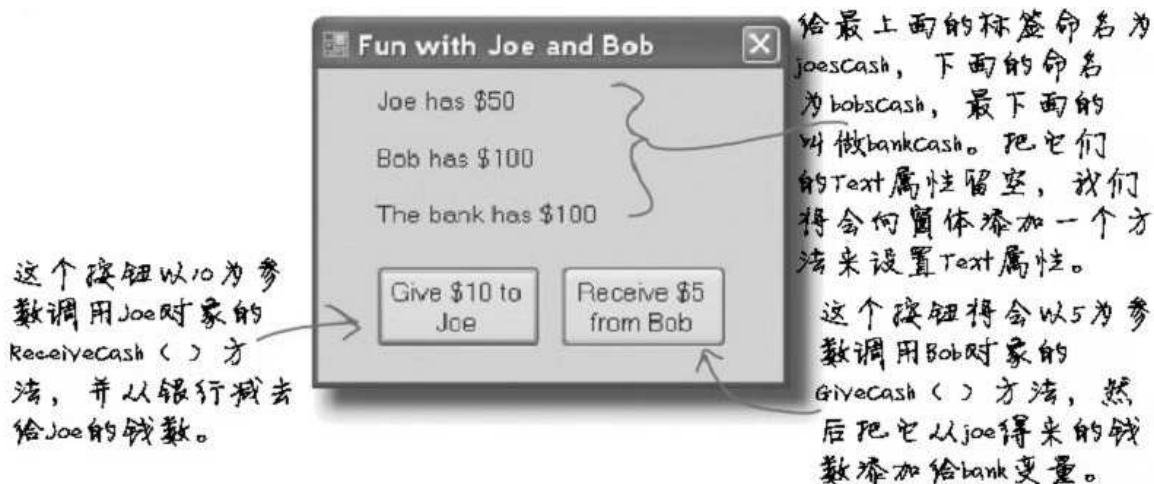
注意大括号。很容易就写错了，每一个左括号要有一个右括号对应。如果匹配正确，IDE 会为你自动缩进。

创建一个窗体来和你的 guy 对象交互

Guy 类很不错，但是这只是一个开头儿。现在创建一个使用两个 Guy 实例的窗体。窗体有显示这个人的名字和有多少钱的标签，还有从他们那儿拿钱或给钱的按钮。

1 向你的窗体添加六个标签和两个按钮

窗体右手边的标签显示人有多少钱。我们还会给窗体添加一个叫做 bank 的变量--右手边的第三个标签显示银行有多少钱。你可以自己给标签命名。你可以单击你想要命名的标签来在属性窗体里面修改 Name 属性。这将会让你的代码易读，因为你的标签名字是“joesCash”，“bobsCash”而不是“label4”，“label5”。



2 向你的窗体添加变量

你的窗体需要两个人，所以需要两个 Guy 类型的变量。命名为 joe 和 bob。然后向窗体添加一个叫做 bank 的变量来存放银行可以给别人或从别人那儿拿来的钱数。

```

namespace Your_Project_Name {
    public partial class Form1 : Form {
        Guy joe;
        Guy bob;
        int bank = 100;

        public Form1() {
            InitializeComponent();
        }
    }
}

```

因为你要用 Guy 的对象来保持 joe 和 bob，所以你要用 Guy 来声明这两个变量。

银行里的钱数因为从 Guy 对象接受和给予他们钱而涨涨跌跌。

3 给窗体添加一个方法来更新标签

窗体右手边的并且显示每个人有多少钱和银行有多少钱。所以给窗体添加一个 UpdateForm 方法来更新它们--确保方法返回值是 void 来告诉 C# 这个方法没有返回值。把这个方法添加到 bank 变量的下面：

```

public void UpdateForm() {
    joesCash.Text = joe.Name + " has $" + joe.Cash;
    bobsCash.Text = bob.Name + " has $" + bob.Cash;
    bankCash.Text = "The bank has $" + bank;
}

```

注意标签是通过用 Guy 对象的名字和 Cash 字段来更新的。

这个新方法很简单。它通过修改标签的 Text 属性来更新标签。每个按钮都通过调用它来更新标签。

4 双击每一个按钮并添加这些代码进去来与对象交互

确保左手边的按钮叫做 button1，右手边的按钮叫做 button2。然后双击两个按钮--IDE 就会添加两个方法分别叫做 button1_Click() 和 button2_Click()。向方法内添加下面的代码：

```

private void button1_Click(object sender, EventArgs e) {
    if (bank >= 10) {
        bank -= joe.ReceiveCash(10);
        UpdateForm();
    } else {
        MessageBox.Show("The bank is out of money.");
    }
}

private void button2_Click(object sender, EventArgs e) {
    bank += bob.GiveCash(5);
    UpdateForm();
}

```

当用户点击 "Give/10 to Joe" 按钮的时候，窗体调用 Joe 对象的 ReceiveCash() 方法--前提是银行有足够的钱。

银行要至少有10元才能给 Joe 钱。如果不够，就会弹出这个消息框。

"b" 按钮不需要检查银行有多少钱，因为它只需要把 Bob 给的钱存入银行。如果 Bob 没钱了。GiveCash() 将会返回 0。

5 一开始给 Joe\$50 给 Bob\$100

你自己来通过设置 Name 和 Cash 属性来使得 Joe 和 Bob 以 50 和 100 美元开始。把它写在 InitializeComponent() 下面。这一句是在窗体初始化的时候只被调用一次的特殊的方法的一部分。做完这些之后，双击按钮几次，确认一个按钮从银行取\$10 并给 Joe，另一个按钮

从 Bob 那儿去\$5 并存入银行。



```
public Form() {
    InitializeComponent();
    // Initialize joe and bob here!
}
```

把初始化两个Guy对象
和设置它们的Name,
Cash字段的代码写在这儿



你自己来通过设置 Name 和 Cash 属性来使得 Joe 和 Bob 以 50 和 100 美元开始。把它写在 InitializeComponent () 下面。

```
public Form1() {
    InitializeComponent();
```

我们在这儿开始设置Guy
的第一个实例。第一行创
建对象，后面两行设置字
段值。

```
bob = new Guy();
bob.Name = "Bob";
bob.Cash = 100;
```

确保要调用UpdateForm ()
，以使得在窗体弹出的时候
标签显示的内容正确。

```
joe = new Guy();
joe.Name = "Joe";
joe.Cash = 50;
```

然后用同样的方法
处理第二个对象

```
UpdateForm();
}
```

确保要把项目
存盘--再过几
页我们还要回转来

there are no
Dumb Questions

问：为什么上面的解答没有以 “Guy bob=new Guy ()” 开始？为什么没有写 “Guy” ？

答：因为在窗体开始处你已经声明过 bob 变量了。还记得 “int i = 5” 这个语句与 “int i” 和 “i = 5” 这两个语句是一样的吗？这儿也一样。你可以在一行中来声明 bob 变量，就像这样：“Guy bob = new Guy ()”。但是你已经写了这个语句的前半部分 (“Guy bob;”) 了。所以现在只需要这一行的后半部分了，就是设置把 bob 变量设置为 Guy 实例的语句。

问：好。那为什么不把窗体开头儿的 “Guy bob;” 这一行删掉呢？

答：那样的话 bob 变量就只会存在于 “public Form1 ()” 这个方法里了。在方法内部声明变量，这个变量只在方法内部有效--在别的方法里面不能访问。但是如果你在方法外的窗体里

或者一个类里声明变量，那么你就可以在该窗体或类的其他方法里面访问这个变量。

问：如果我在方法内也写一个“Guy”会怎么样？

答：那就麻烦了--你的窗体无法工作，因为窗体的 bob 变量根本就没被赋值。考虑一下，你就知道为什么会这样了。如果你的窗体开头儿这样写：

```
Public partial class Form1:Form { Guy bob;
```

然后在方法里这样写：

```
Guy bob = new Guy();
```

这样你就声明了两个同名的变量了，这有点混乱。一个在整个窗体中可访问，另一个--在方法内部添加的那个--只在方法内部可访问。下一行 (bob.Name = “Bob”;) 只会更新局部变量，不会触及窗体级的变量。所以运行的时候，你会收到一个讨厌的错误

(“NullReferenceException not handled”), 意思就是说你尝试在给变量用 new 语句赋值之前使用它。

还有一个更简单的方法来初始化对象

基本是你创建的每一个对象都需要被以某种方式来初始化。Guy 对象也不例外--在你设置它的 Name 和 Cash 字段之前这个对象是没用处的。因为初始化字段值太普遍了，C# 给你提供了一个简捷方式，叫做对象初始化器。IDE 的智能感应会帮你完成它。



对象初始化器只在 **C# 3.0** 中可用。

如果你在用 VS2005，那不行。下载 VS 2008 Express Edition 吧--它是免费的，它可以和你已经安装的 VS 2005 共存。

1 原本你是这么写代码来初始化 Joe Guy 对象的。

```
joe = new Guy();
joe.Name = "Joe";
joe.Cash = 50;
```

2 删除后两行，还有“Guy()”后面的分号也删掉，再添加一个左大括号。

```
joe = new Guy() {
```

3 敲空格。IDE 马上就会弹出一个智能感应窗口，给出所有你可以初始化的字段。

```
joe = new Guy() {
```



4 按 tab 键来添加 Cash 字段。然后设置它为 50.

```
joe = new Guy() { Cash = 50
```

5 敲一个逗号。马上，就会显示其他的字段。

```
joe = new Guy() { Cash = 50,
```



6 结束掉对象初始化器。现在，你省了两行代码！

```
joe = new Guy() { Cash = 50, Name = "Joe" };
```

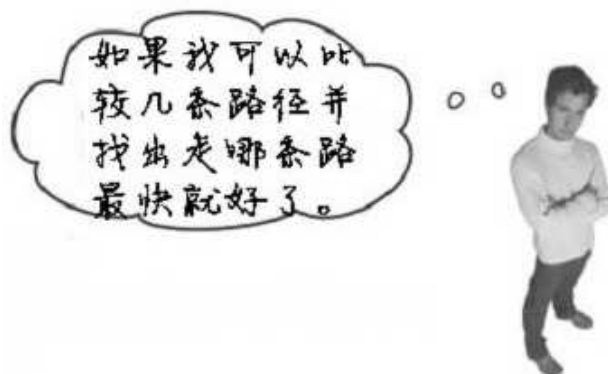
这个新的声明和原来写的三行代码作用是一样的。区别就是现在短一点，还易读一点。

对象初始化器为你节省时间，并使得你的代码紧凑易读...而且 **IDE** 会帮你来写。

设计直观的类的几个好主意

* 你构建程序是为了解决问题

花点时间考虑一下要解决得问题。问题容易分解为多个小部分吗？你会怎么给别人描述这个问题？设计类的时候需要考虑很多事儿。



* 你的程序会用到什么现实世界的东西？

一个帮助动物园管理员管理动物的喂食的程序可能会含有代表不同种类食物或不同种类动物的类。



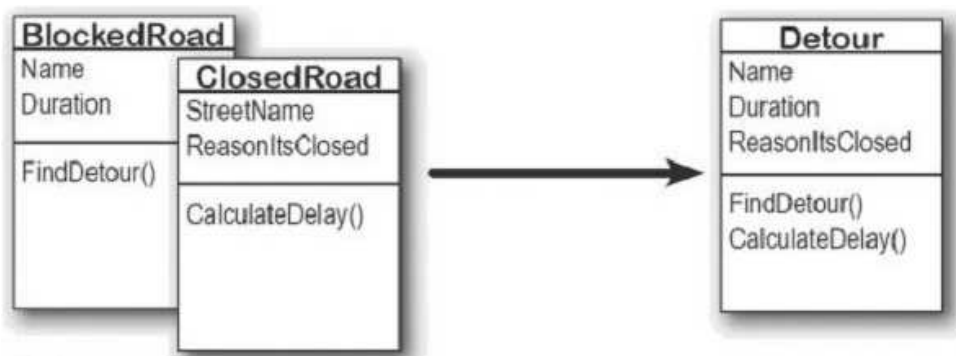
* 给类和方法命名要直观

要让别人一看你的类和方法的名字就知道类和方法是做什么的。



* 寻找类之间的共同点

如果两个类很相似的话，它们可能可以合并为一个类。糖果制造系统可能有三四个涡轮，但是只有一个方法来关闭速闭阀，这个方法取一个涡轮号作为参数。



向“Fun with Joe and Bob”程序添加按钮来让两个人给彼此钱。

1 用对象初始化器来初始化 Guy 的对象 Bob

你已经在 Joe 身上这么做过。现在让 Bob 实例也用对象初始化器创建。

如果你已经点击过按钮了，把它删了，再添加回窗体，并重命名。然后把IDE之前创建的 `buttons_Click()` 方法删除，使用IDE现在添加的方法。

2 再向窗体添加两个按钮

第一个按钮让 Joe 给 Bob 10 美元，第二个按钮让 Bob 给 Joe 5 美元。在双击按钮之前，去属性窗口通过 Name 属性来修改每个按钮的名字--它在属性窗口窗口的顶部。给第一个按钮命名为 `joeGivesToBob`，第二个按钮命名为 `bobGivesToJoe`。

这个按钮让Joe给Bob 10美元，所以你应该在属性窗口中用Name属性把按钮命名为joeGivesToBob。

这个按钮让Bob给Joe 5美元。命名它为bobGivesToJoe。

3 让按钮可以工作

在窗体设计器里面双击 joeGivesToBob。IDE 将会向窗体添加一个 joeGivesToBob_Click(), 它在每次按钮被点击的时候都会运行。填写这个方法让 Joe 给 Bob 10 美元。然后双击另一个按钮，并填写 IDE 创建的 bobGivesToJoe_Click()方法来让 Bob 给 Joe 5 美元。确保窗体在每次现金转手之后都会更新自己。



Exercise

向“Fun with Joe and Bob”程序添加按钮来让两个人给彼此钱。

```
public partial class Form1 : Form {
    Guy joe;
    Guy bob;
    int bank = 100;

    public Form1() {
        InitializeComponent();
        bob = new Guy() { Cash = 100, Name = "Bob" };
        joe = new Guy() { Cash = 50, Name = "Joe" };
        UpdateForm();
    }
}
```

这儿是两个Guy类的实例的对象初始化器。Bob以他的名字和100美圆被初始化。

```

public void UpdateForm() {
    joesCash.Text = joe.Name + " has $" + joe.Cash;
    bobsCash.Text = bob.Name + " has $" + bob.Cash;
    bankCash.Text = "The bank has $" + bank;
}

private void button1_Click(object sender, EventArgs e)
{
    if (bank >= 10) {
        bank -= joe.ReceiveCash(10);
        UpdateForm();
    } else {
        MessageBox.Show("The bank is out of money.");
    }
}

private void button2_Click(object sender, EventArgs e)
{
    bank += bob.GiveCash(5);
    UpdateForm();
}

private void joeGivesToBob_Click(object sender, EventArgs e) {
    bob.ReceiveCash(joe.GiveCash(10));
    UpdateForm();
}

private void bobGivesToJoe_Click(object sender, EventArgs e) {
    joe.ReceiveCash(bob.GiveCash(5));
    UpdateForm();
}
}

```

让Joe给Bob钱，我们要调用Joe的GiveCash()方法并把这个方法的返回值传给Bob的ReceiveCash()方法。

仔细看一下Guy的方法是怎么被调用的。GiveCash()方法的返回值被当做参数传递给ReceiveCash()方法。

这儿关键就是考虑清楚谁掏钱出来，谁拿钱回去。

Pool Puzzle



你的任务是吧游泳池里面的代码片段取出来放进这儿的代码的空行里。同一个代码段可以用多次，但是没必要把所有的代码段都用到。你的目的是让类可以编译并运行并产生列出来的输出结果。


```

public partial class Form1 : Form
{
    private void button1_Click(object sender, EventArgs e)
    {
        String result = "";
        Echo e1 = new Echo();
        Echo e2 = new Echo();
        int x = 0;
        while ( x < 4 ) {
            result = result + e1.hello() + "\n";
            e1.count = e1.count + 1;
            if ( x == 3 ) {
                e2.count = e2.count + 1;
            }
            if ( x > 0 ) {
                e2.count = e2.count + e1.count;
            }
            x = x + 1;
        }
        MessageBox.Show(result + "Count: " + e2.count);
    }

    public class Echo {
        public int count = 0;
        public string hello() {
            return "helloooo...";
        }
    }
}

```

这是正确答案。
下面是附加题答案！
Echo e2 = e1;