

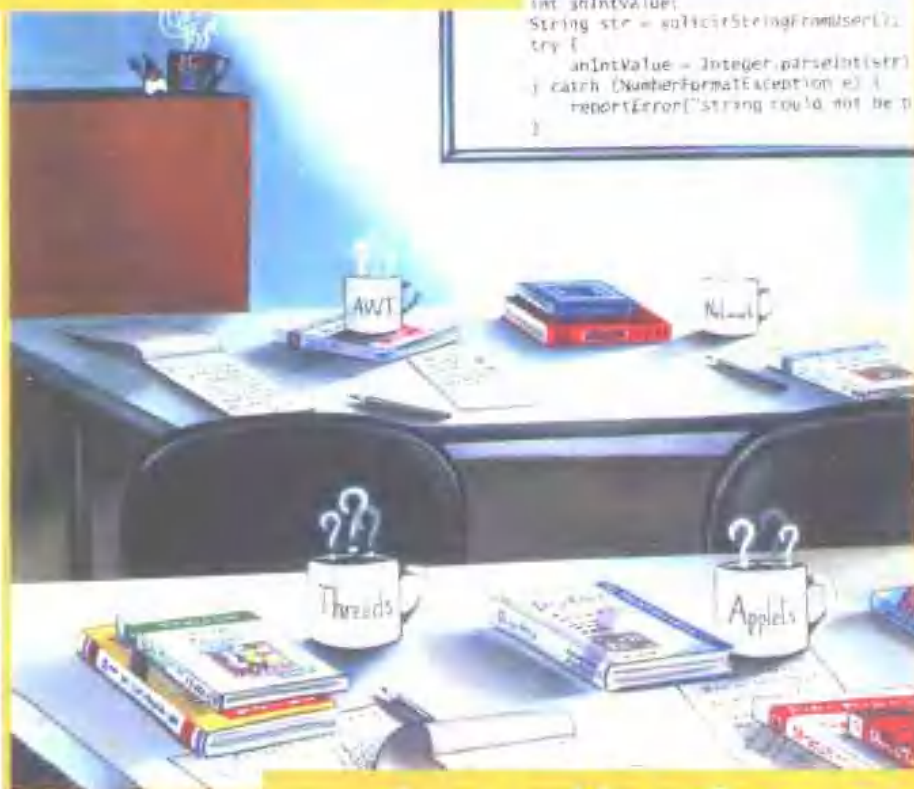
The Java FAQ

Java 经典问答

The Java Series

Jonni Kanerva 著
陈霞 译

```
int getIntValue()  
String str = solicitorStringFromUser();  
try {  
    intIntValue = Integer.parseInt(str);  
} catch (NumberFormatException e) {  
    reportError("String could not be p  
}
```



...from the Source™

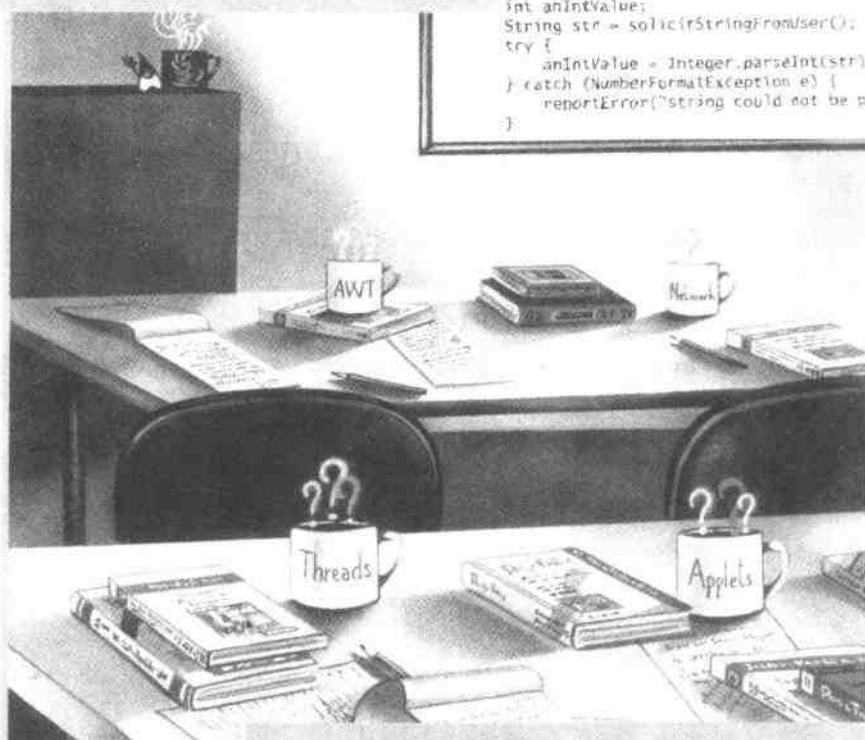


The Java FAQ

Java 经典问答

Jonni Kanerva 著
陈霞 译

The Java Series



...from the Source

中国电力出版社

内 容 提 要

Java 系列丛书是关于 Java 的最完整、专业和指定的官方教材,它们直接来自 Sun Microsystems 公司的 Java 技术创始人。本书由 Sun JavaSoft 工作组的成员编写,提供了大量翔实的信息,它们都是您在学习 Java 过程中需要理解,以便可以用它迅速开发出灵活、健壮、方便、安全的 Java 程序和 Internet Applet 所必须的。Java 系列丛书是 Java 学习者不可缺少的参考资料

本书适合软件设计开发人员及大专院校师生阅读。

图书在版编目(CIP)数据

Java 经典问答/(美)强尼埃诺编著;陈霞译.-北京:中国电力出版社,2001.12

ISBN 7-5083-0841-7

I. J… II. ①强…②陈… III. Java 语言-程序设计-问答
IV. TP312.44

中国版本图书馆 CIP 数据核字(2001)第 077983 号

北京版权局著作权登记号 图字 01-2001-2224

本书英文版原名: The Java FAQ

Published by arrangement with Addison Wesley Longman, Inc.

All rights reserved.

本书中文版由美国培生集团授权出版,版权所有。

中国电力出版社出版、发行

(北京三里河路 6 号 100044 <http://www.infopower.com.cn>)

实验小学印刷厂印刷

各地新华书店经售

2002 年 1 月第一版 2002 年 1 月北京第一次印刷

787 毫米×1092 毫米 16 开本 17 印张 384 千字

定价 28.00 元

版 权 所 有 翻 印 必 究

(本书如有印装质量问题,我社发行部负责退换)

译者序

Java 技术在极短的时间内就取得了令人瞩目的成就，它以平台无关性、动态性、安全性、强壮性以及全面支持 Internet 技术等成为目前最流行的跨平台应用系统开发工具，Java 语言的出现为我们带来了前所未有的活力和广阔的开发空间。

本书是一本专门讲解 Java 语言基础知识和基本概念的优秀书籍，它采用独特的“问题—解答”形式，不仅包括了将近 200 个在 Java 学习和应用中最重要、信息涵盖量最大的问题，并针对每个问题和案例都给出了准确的解释和建设性解决方案。本书的所有问题都直接来源于 java@java.sun.com 上用户的反馈信息，所有的解答都出自 JavaSoft 的 Java 语言设计者们，从而保证了本书的正确性和高质量。

本书包括了对象、类、方法、接口、例外、数组、虚拟机、Applet、AWT 组件和容器、布局、事件、图形、线程、SOCKET 等 Java 主题，特别值得一提的是作者还采用不同 JDK 版本在一些问题的不同解决方案上的对比，向大家展示出了 Java 虚拟机 JDK 的不断扩充完善的过程。本书是学习 Java 语言程序设计技巧、培养良好编程风格以及了解 Java 开发内幕的一部非常好的书籍。此外本书采用的分类编号方式又有助于问题的定位查找，实在是各层次 Java 学习者（特别是初学者）手头必备的一本参考书。

由于时间仓促和本人水平有限，翻译过程中难免会出现一些错误和纰漏，敬请广大读者给予批评指正！

译者
2001 年 11 月

序

Java™ 技术在极短的时间内就取得了令人瞩目的成就，它之所以能在如此短的时间内被人们所接受，从某种意义上说反映了它技术上的优势，即我们常说的：平台无关性（Platform-neutrality）、动态性（Dynamic）、面向对象（Object-Oriented）、垃圾收集（Garbage collection）、安全性、并发处理（Concurrency）、强壮性以及全面支持 Internet 技术等，Java 语言已成为一种成熟的、可用来解决各种一般问题的编程语言，它所提供的标准类库更是为开发各种应用提供了一个广阔的空间。

然而从许多开发者对 Java 技术接受的速度和寄予的热情上表明，除了技术上的完备性以外，使用 Java 的确是一件令人愉快的事，它省去了应用程序开发过程中令人讨厌并且容易犯错的诸如内存管理、跨平台接口等内容，使开发者的全部精力都可以集中在程序功能的实现上面，语言本身也相对简单易学。在这一点上，我们应该感谢 Java 开发时所确定的必须遵循的一条基本原则——简单，而这条原则的结果就是——好学。Java 不仅使用起来很有趣，就是学起来也非常有意思。

这本书就是为 Java 初学者而写的，它的目标就是简单实用：

- 对大量相关知识提供了一个简便的查找方式
- 对学习和应用中常见的错误提供了分析和解决方法
- 给出了开发应用程序方面的基本知识

本书不是一本 Java 技术的详细论文，对于这一点，如果有必要可以去查阅有关 Java 语言定义、Java 虚拟机或 Java API 等方面的书籍；它也不是一步一步学习 Java 的指导材料，对于这一点，可以去看有关 Java 程序设计方面的教程。

本书充分利用了教师的两大法宝：问题和示例。问题确定了需求：如果您可以指出您缺乏的东西，那您就离得到它不远了，问题将新的信息和已有的知识紧密地联系在一起。

同样，例子为更好地理解问题提供了一个入口。一个例子比直接揭示知识的本身更容易被理解和接受。而且实例代码也很有用，它通常可以被引用在一个更大的实际程序设计中。

什么样的问题更具代表性呢？本书所采用的问题大多来自 java@java.sun.com 这一 E-mail 地址所搜集的问题。这个 E-mail 地址是 SUN 公司为 Java 技术提供技术支持的，特别是 Java 和 HotJava 从最初的 1.0 α 版，到 β 版，一直到第一个用户版。尽管本书的书名是《Java 经典问答》，但实际上我们所指的问题是指那些为了学好 Java 而不得不解决的疑难问题。

当提出问题的时候，有必要先了解一下问题是否已经被解答过了，本书实际上是《C++

经典问答》的姊妹篇：也是在每一个问题后立刻给出一个简洁的答案，接着再展开并给予例示。

许多解答都是以一个 URL 结束的，它实际上是提供了与问题相关的实例代码、小程序或其他的一些资料。可以通过网址 <http://java.sun.com/Series> 访问这些内容。

《Java 经典问答》覆盖了 Java 语言、Java 虚拟机和 `java.lang`、`java.applet`、`java.awt.image`、`java.awt.event`、`java.io`、`java.net` 和 `java.util` 包中的标准类。这些部分都是 Java 开发工具 JDK 1.0 和 JDK 1.1 包含的基本内容，而与其他版本（如 JDK 1.0.2 和 JDK 1.1）有关内容和源代码也在本书中做了明确的标注。本书不包括 JDK 1.1 版本中新添加功能包 `java.rmi`、`java.beans` 和 `java.security` 等方面的内容。

如果没有 JavaSoft 和其他专业人士和个人的帮助，这本书是不可能出版的。Lisa 首先酝酿了这本书并鼓励我完成它，作为这一系列出版物的编辑，她耐心而恒久的关心使我可以坚持将这一庞大的工程完成。此外，尽管其他的工作已经很多了，Carla Schroer 仍然允许我接受这项工作。Mary Aline 对前一些章节草稿的仔细修改提醒我尽量保持书写的清楚、简洁和语法顺畅。Mary Campione、Eric Chu、Jazmin Ellis、Doug Kramer、Nancy Lee、Rick Levenson、Marvin Ma、Ron Mandel、Freeman Murray、Scott Rautmann、Richard Scorer、Greg Voss 以及 Kathy Walrath 给我的工作环境增加了许多生活气息，提高了我工作和休息的质量。我要感谢 Tom Ball、Dave Brown、Patrick Chan、Dave Connelly、Jeff Dinkins、Pavani Diwanji、Amy Fowler、Jim Graham、Jeannette Hung、Herb Jellinek、Tim Lindholm、Marianne Mueller、Tim Prinzing、John Rose、Georges Saab、Nakul Saraiya、Sami Shaio、Arthur van Hoff、Chris Warth 和 Frank Yellin 的帮助，和这些天才工程师们工作在一起的确是一件令人愉快的事，从他们身上我学到了许多东西。Addison Wesley、Mike Hendrickson、Katie Duffy、Marina Lang 和 Simone Payment 使得本书开发进程中的困难部分都很容易地被解决了，他们的幽默、努力工作以及谦恭的敬业精神完全超出了他们的职责。Addison Wesley 的初稿审阅以及核查人员给本书提出了详细的建议和建设性的批评。在下一阶段的工作中，Victi Hochstedler 帮助我将这本书制作成最后的格式，Rosemary Simpson 为本书加了索引。

此外，我还要感谢我的家庭，他们理解这项普通工作，并为此放弃了许多东西。我的妻子 Jean 像一块宝石，她的智慧、诚实、体贴和处理生活问题的才干是我生活和 15 年工作的强大支柱，我希望本书不仅仅拥有内容，更拥有一种精神。此外我的孩子 Kirti、Silva 和 Justin 给我的生活带来了无穷的乐趣，为了能有更多的时间和他们在一起是激发我将这个工程尽快完成的一个特殊的动力。家人们也有一个经常问的问题——“书什么时候才可以完成”？——我一直无法回答，现在答案就在这里。

Jonni Kanerva
Mountain View, 加利福尼亚

目 录

译者序

序

第 1 章	类、接口和包	1
	对象、类和方法	1
	子类、重载和覆盖	12
	接口和抽象类	23
	包和访问修饰符	31
第 2 章	Java 语言	37
	常量和表达式	37
	变量和方法	49
	数组	61
	例外	64
第 3 章	Java 虚拟机	71
	虚拟机	71
第 4 章	Applet	83
	Applet 和应用程序	83
	安装 Applet	87
	Applet 用户界面	95
	Applet 程序结构	100
	Applet 通信	104
	其他	107
第 5 章	抽象窗口工具集	110
	组件、容器和对等	110
	窗口、框架和对话框	119

📖 其他.....	124
第 6 章 事件——JDK 1.0.2	127
📖 事件.....	127
第 7 章 事件——JDK 1.1	141
📖 事件类、监听和方法.....	141
📖 语义事件.....	151
📖 低级事件.....	157
第 8 章 图形	163
📖 绘制 AWT 组件.....	163
📖 加载和画图.....	169
📖 图形——JDK 1.0.2	182
📖 图形——JDK 1.1	185
第 9 章 线程	191
📖 创建和控制线程.....	191
📖 线程的交互.....	198
📖 用户线程和系统线程.....	210
第 10 章 输入、输出和网络.....	213
📖 基本输入和输出.....	213
📖 URL 连接.....	224
📖 Internet 地址	233
📖 Socket.....	235
第 11 章 其他	245
📖 java.lang 和 java.util 中的类.....	245
📖 音频.....	251
📖 其他.....	255

类、接口和包

对象、类和方法 (Q1.1~Q1.10)

子类、重载和覆盖 (Q1.20~Q1.28)

接口和抽象类 (Q1.29~Q1.30)

包和访问修饰符 (Q1.29~Q1.34)

对象、类和方法

Q1.1

什么是对象?

一个对象就是一个程序单元，它将一组结构化数据和对这些数据进行的各种操作结合在一起。

面向对象程序设计将我们熟悉的非面向对象程序设计语言（如 C 语言）中的函数与数据之间的关系倒了过来。在 C、Pascal 或 Basic 中，程序常常是建立在对数据的直接处理之

上：首先要确定数据结构，然后给出可以从程序的任何地方对这些数据进行处理（如查询或改变其值）的函数。因此与数据结构有关的内容会出现在程序的各个角落，如果要改变程序某一部分中的数据结构，势必会因此而影响到程序的其他部分。

面向对象程序设计方法恰恰为降低这种程序代码之间的相互依赖提供了标准的工具和技术解决方案：一个对象由一组结构化数据的定义开始，并且还包含了对这些数据进行的所有可能的存取操作。在对象中，与数据结构有关的所有操作都和这个结构紧密地结合在一起，而不是被分散在程序的各个角落。

类（Q1.2）实际上是具有相同性质的一类对象的集合，它同时定义了对象拥有的数据和操作这些数据的一系列方法（Q1.3）。我们在程序设计中接触的每一个对象都隶属某个类并被称为那个类的实例（Q1.7）。

面向对象的程序设计将操作和它们的数据紧密地结合在一起，从而使我们可以通过向对象请求服务而不是直接操作它的数据来完成对数据的存取。这种小小的不同可以使申请服务方从服务提供方那得到极大的好处：请求方（程序的某一部分）只需知道请求什么样的服务就可以了，而其他的事情则全部由执行方（对象）负责完成。现在，只有部分程序是相互依赖的，不是通过数据结构，而是通过该对象声明所能完成的功能。

Java™ 是面向对象的程序设计语言，这就意味着类、实例、方法构成了整个程序设计的核心内容。

请参看：Q1.2、Q1.3、Q1.7

Q1.2

什么是类？

类是对象的设计蓝图：它根据对象持有的数据和对数据进行的操作的特点而定义类对象的设计模版。

类是面向对象程序设计中最基本的单元，它定义了类对象的设计模版：不仅为对象确定了一个数据结构，而且还为存取这些数据定义了一系列操作。类通常也是制造对象的工厂，当程序运行时，可以通过类来创建一个或多个对象——我们把它称为类的“实例”（Q1.7）。类的每个实例都拥有它自己的数据，因此，我们可以修改一个实例的某些内容而保持其他实例不变。类只是详细定义了对象所具有的各种功能，而具体的实现则由对象本身去完成。

例如 Java 中的 String 类就是为所有字符串定义的一个对象类型。一个 String 实例由一组 Unicode 字符序列和可以从该字符序列中提取有关信息的方法 (Q1.3) 组成。尽管只有一个 String 类, 但在一个 Java 程序中却可以拥有许多 String 类的实例。

Java 中类的声明看起来更像 C 语言中的结构定义, 它至少要包括一个关键字 “Class”、一个类名、一对花括号的开始 “[” 和结束 “]”。此外, 类声明中还包括许多可选部分如: 访问修饰符 (Q1.32)、父类定义 (Q1.12)、接口定义 (Q1.20) 以及类体中的其他内容等等。

下面的例子定义了一个只有一个数据 (又称域) 和一个方法的类:

```
class SimpleClass{
    int count; //field
    void incrementCount(){
        ++count;
    }
}
```

为学好面向对象程序设计方法, 我们必须要充分理解类与实例这两个概念上的区别和联系, 表 1.1 通过生活中的一些常识为我们类比了这两个概念的不同之处。

表 1.1 类和实例

类	实 例
小甜点的烹饪法和刀具	小甜点
建房方案	建好的房子
橡皮图章	印出的图案
底片	洗好的照片

注意

从现在开始, 本书将按照 JLS 《Java 语言规范》(第 38 页) 中所使用的 “对象” 这一术语来指代我们将要讨论的所有类的实例和数组 (Q2.19)。虽然这种叫法可能需要一点时间来适应, 但是它在意思表达上却更加清楚和一致。

请参看: Q1.1、Q1.3、Q1.7、Q1.20、Q1.32、Q2.19

Q1.3

什么是方法？

在面向对象程序设计语言中，方法是体现对象功能的最基本的可执行程序设计单元，它隶属于某一个类，并且可以被特定的对象和类本身调用。

方法相当于非面向对象语言中的函数或子程序，它和函数一样有以下几个组成部分：

- 名字
- 一个可以为空的输入参数及其类型名表
- 一个可选的返回类型（关键字 `Void` 表示方法没有返回类型）
- 可执行代码体

下面的程序片段声明了一个方法，方法的输出结果是两个输入值（`Double` 型）的平均值（`Double` 型）：

```
double average(double a, double b) {  
    return(a+b)/2.0;  
}
```

因为方法是隶属于某些类（Q1.2）的，所以只可以在该类的定义中定义您的方法。值得注意的是，本书为方便起见除特殊情况，源程序代码片段都没有包括类定义部分。

通常可以通过方法来检查或处理对象中所存储的数据，例如，Java 的 `String` 类，它为确定一个字符串实例的长度提供了一个叫 `length()` 的方法，它没有参数，返回值是代表调用它的字符串实例长度的 `int` 类型的数，请看下面的例子：

```
String aString="Bicycle Shop Quarterly News";  
int howLong=aString.length();
```

方法不是简单地被激活，而是通过一个确定的对象（或类 Q1.6）来调用。让我们再思考一下下面的程序片段，它创建了两个 AWT——抽象窗口工具集（`Abstract Window Toolkit`）——按钮实例并且为每个按钮设置了要显示的文本内容：

```
/* in Java: */  
Button button1=new Button();  
Button button2=new Button();
```

```
button1.setText("Push Me First");
button2.setText("Push Me Second");
```

`button1` 调用 `setText` 方法就等于让 `Button` 类去查找关于 `setText` 的定义, 然后使用 `button1` 中的有关数据去执行方法体中的代码。在 `button2` 上调用 `setText` 方法导致同样的结果, 只是同样的程序片段是作用于 `button2` 而不是 `button1`。

在非面向对象语言如 C 中, 您可能会使用一个类似的 `setButtonText` 函数并把按钮作为其中的一个参数:

```
/* in C: */
Struct Button *button1=makeButton();
Struct Button *button2=makeButton();
SetButtonText(button1,"Push Me First");
SetButtonText(button2,"Push Me Second");
```

(关于 C 中的结构和 Java 中类之间关系的讨论可以参看问题 Q2.10。)

因为每个类只负责自己的方法, 所以不同类可以使用具有相同名字、相同参数和返回类型的方法。例如, AWT 中 `Label` 类也定义了一个叫 `setText` 的方法, 参数是一个 `String` 类型字符串:

```
/* using Label's setText method rather than Button's: */
Label aLabel=new Label();
aLabel.setText("Read me—I'm a label.");
```

在这个例子中, `Label` 类提供了它自己对 `setText` 的定义, 这个定义完全不同于 `Button` 类给出的定义。

从调用者的角度看, 虽然方法不同, 但它们的作用看起来却非常相似, 这也是面向对象程序设计的一个关键部分: 它对动作的请求和请求是如何完成的细节完全分离开。只要对象的方法名字、参数表和返回类型均相同 (Q1.14), 就可以直接调用该方法而无需知道对象到底属于哪个类。而到底是哪段程序被执行则由目标对象的类来决定, 而不是调用者。

请参看: Q1.2、Q1.4、Q1.6、Q1.14、Q2.10

Q1.4

什么是方法的声明？

一个方法的声明是由方法名和方法输入参数（包括个数以及它们的类型）组成。

一个方法的声明足以表明该方法在所属类中的惟一性。换句话说，当一个类要从方法调用映射到相应的可执行方法体时，方法声明成了一个独一无二的查找键。

下面是 AWT 中 `Rectangle` 类的方法声明：

```
a. isEmpty()
b. intersection(Rectangle)
c. setLocation(int,int)
d. contains(int,int)
e. contains(Point)
```

（请注意，在这里我们引用的方法声明并不是合法的 Java 源代码，它们只代表方法声明本身。）a、b、c 三个声明在名称、参数个数和类型上都各不相同；c 和 d 只在名字上不同；d 和 e 虽然在名字上相同，但在参数个数和类型上却不相同。

在这里大家有必要记住，一个方法的输出——返回值或声明的例外——对方法的声明并没有多大的影响，尽管编译器会利用这些信息来检查方法是否被覆盖(Overriding)（Q1.14、Q2.24）

请参看：Q1.13、Q1.14、Q2.24

Q1.5

实例变量和类变量有什么不同？

一个实例变量代表的是类定义中的一个独立的数据项，类的每个实例中都拥有一个属于自己的拷贝；而一个类变量则代表的是可以被这个类和它所有的实例共享的一个数据项。

一个实例变量代表的数据表示每个实例都有它自己的一份拷贝。实例变量相当于 C 或

C++中结构类型的某个域。比如，如果程序有 20 个 **Button** 按钮实例，每个实例都有它自己的 **label** 标签，实例变量为按钮保存标签上的字符串信息。

相反，类变量则属于类本身，该类的所有实例访问的都是该变量的同一个拷贝。因此，一个类变量可以充当一个共享资源和可以提供实例之间通信的一种方式。类变量通常用关键字 **static** 声明。

下面的程序段声明了一个有两个实例变量和一个类变量的类：

```
class Example {
    int i1; // no static keyword, therefore instance variable
    String s;
    Static int i2; // static keyword, therefore class variable
}
```

请参看：Q1.6

Q1.6

实例方法和类方法有什么不同？

实例方法有“**this**”引用，而类方法没有

实例方法总是和类实例一起出现。一个实例方法必须被一个特定的类实例调用，它提供了访问该实例中各种信息的一种独特方式。例如：

```
String s1="abcde" ; // one String instance
String s2="fgh"; // another String instance
int i1=s1.length(); // i1=5
int i2=s2.length(); // i2=3
```

length 这个方法返回的是目标对象（如 **s1**, **s2**）中的字符个数，就好像那个目标是 **length** 函数的一个参数（C 语言中的 **cf.strlen (aString)**）。但定义一个实例方法时，可以使用“**this**”这个关键字来指代目标实例。

类方法没有目标实例，所以不需要“**this**”引用。可以随时随地调用类方法，即使还没有创建该类的实例。类方法如同类变量一样，在声明的时候需加上关键字“**static**”。

例如，Thread 线程类中的 `currentThread` 方法就是一个类方法。可以通过 Thread 类本身来调用它，它确定当前正在执行的线程并返回对该线程的一个引用。

```
Thread activeThread=Thread.currentThread();
```

现在有许多 Java 程序员更喜欢用名词对象、域、静态域、方法、静态方法来代替实例变量、实例方法、类变量、类方法。表 1.2 简要给出了这些名词之间的联系。

表 1.2 类中各元素命名

成 员	域	实例变量（或非静态域）
		类变量（或静态域）
方 法	方 法	实例方法（或非静态方法）
		类方法（或静态方法）

注意

术语“域”在区别类中定义的变量和方法中的变量时非常有用。

VariableExample.html

请参看：Q1.2、Q1.3、Q1.5

Q1.7

如何创建类实例？

创建类实例的方法通常是调用该类的构造函数。

在 Java 中，每当创建对象（如实例和数组）时系统都会从它管理的内存中分配一定的空间给这个对象。虽然您永远无法直接访问那些堆栈，但是 Java 语言和 Java 虚拟机会通过关键字 `new` 和 `newInstance` 方法以及自动垃圾收集器为您处理这一切。

创建实例的标准方法是用关键字 `new`，然后加上类名以及类的某个构造函数所需要的参数，例如：

```
Button myButton = new Button (" Press Me" );
```

这行代码声明了一个叫 `myButton` 的变量，创建了一个叫做“Press Me”的新按钮，并将 `myButton` 指向这个按钮。Java 常用的编码风格是将类变量的声明和初始化放在同一个程

序行中，也可以简单地声明一个 `Button` 类型的实例变量，如

```
Button myButton;
```

这样就创建了一个叫做 `myButton` 的变量，它被缺省初始化（参看 Q1.10）为 `null`；或者如果声明的是一个本地变量（即方法内部的变量），系统会要求给它先赋初值后才可以使用。

还可以通过调用来创建或得到类的实例。有些方法的返回值通常会是一个新的实例，例如 `Class` 类（Q2.15、Q3.7）中的 `newInstance`、`Integer` 类（Q10.4）中的 `valueOf`；此外，一些方法如 `InetAddress` 的 `getByName`（Q10.16）也能返回一个实例，只是返回的实例与原来的相同。

请参看：Q1.10、Q2.15、Q3.5、Q3.7、Q10.4、Q10.16

Q1.8

什么是抽象方法？

一个抽象方法包括了定义此方法的各方面内容，只是不包括实现方法的程序代码，即方法体。

一个抽象的方法声明了方法的名字、参数类型、返回值以及例外事件等，只是没有提供方法的实现方案。可以通过加关键字 `abstract` 并在方法体的位置上用一个分号代替来声明这种抽象方法。如：

```
public abstract void drawFigure(); // abstract method declaration
```

抽象方法允许您设计一个类而把部分或所有的代码实现留给子类去完成。包含有抽象方法的类被称做抽象类，但也必须在声明时加上关键字 `abstract`。一旦类中包含了抽象的方法，所有继承此类的子类将不得不为这些方法加上具体的程序代码以实现方法定义的所有功能。

例如，`java.lang` 包中的 `Number` 类就全部是由抽象方法组成的，完整的类定义如下：

```
/* in Number.java (JDK 1.0.2 and 1.1) */
public abstract class Number {
    public abstract int intValue();
    public abstract long longValue();
}
```