定理 1.4

递归的整数打印算法对 $n \geqslant 0$ 是正确的。

证明(通过对 n 所含数字的个数，用归纳法证明之)：

首先，如果 n 只有一位数字，那么程序显然是正确的，因为它只是调用一次 printDigit。然后，设 printOut 对所有 k 个或更少位数的数均能正常工作。我们知道 $k+1$ 位数字的数可以通过其前 k 位数字后跟一位最低位数字来表示。但是前 k 位数字形成的数恰好是$\lfloor n/10 \rfloor$，由归纳假设它能够被正确地打印出来，而最后的一位数字是 $n \bmod 10$，因此该程序能够正确打印出任意 $k+1$位数字的数。于是，根据归纳法，所有的数都能被正确地打印出来。 ■

这个证明看起来可能有些奇怪，但它实际上相当于是算法的描述。证明阐述的是在设计递归程序时，同一问题的所有较小实例均可以假设运行正确，递归程序只需要把这些较小问题的解(它们通过递归奇迹般地得到)结合起来形成现行问题的解。其数学根据则是归纳法的证明。由此，我们给出递归的第三个法则：

设计法则(design rule)。假设所有的递归调用都能运行。

这是一条重要的法则，因为它意味着，当设计递归程序时一般没有必要知道簿记管理的细节，你不必试图追踪大量的递归调用。追踪具体的递归调用的序列常常是非常困难的。当然，在许多情况下，这正是使用递归好处的体现，因为计算机能够算出复杂的细节。

递归的主要问题是隐含的簿记开销。虽然这些开销几乎总是合理的(因为递归程序不仅简化了算法设计而且也有助于给出更加简洁的代码)，但是递归绝不应该作为简单 for 循环的代替物。我们将在 3.6 节更仔细地讨论递归涉及的系统开销。

当编写递归例程时，关键是要牢记递归的四条基本法则：

1. 基准情形。必须总要有某些基准情形，它无需递归就能解出。

2. 不断推进。对于那些需要递归求解的情形，每一次递归调用都必须要使状况朝向一种基准情形推进。

3. 设计法则。假设所有的递归调用都能运行。

4. 合成效益法则(compound interest rule)。在求解一个问题的同一实例时，切勿在不同的递归调用中做重复性的工作。

第四条法则(连同它的名称一起)将在后面的章节证明是合理的。使用递归计算诸如斐波那契数之类简单数学函数的值的想法一般来说不是一个好主意，其道理正是根据第四条法则。只要在头脑中记住这些法则，递归程序设计就应该是简单明了的。

1.4 实现泛型特性构件 pre－Java 5

面向对象的一个重要目标是对代码重用的支持。支持这个目标的一个重要的机制就是泛型机制(generic mechanism)：如果除去对象的基本类型外，实现方法是相同的，那么我们就可以用泛型实现(generic implementation)来描述这种基本的功能。例如，可以编写一个方法，将由一些项组成的数组排序；方法的逻辑关系与被排序的对象的类型无关，此时可以使用泛型方法。

与许多新的语言(例如 C++，它使用模板来实现泛型编程)不同，在 1.5 版以前，Java 并不直接支持泛型实现，泛型编程的实现是通过使用继承的一些基本概念来完成的。本节描述在 Java 中如何使用继承的基本原则来实现一些泛型方法和类。

Sun 公司在 2001 年是把对泛型方法和类的直接支持作为未来的语言增强剂来宣布的。后来，终于在 2004 年末发表了 Java 5 并提供了对泛型方法和类的支持。然而，使用泛型类需要理解 pre－Java 5对泛型编程的语言特性。因此，对继承如何用来实现泛型程序的理解是根本的关键，甚至在 Java 5 中仍然如此。