# 第六章 方法

## 6.1引言

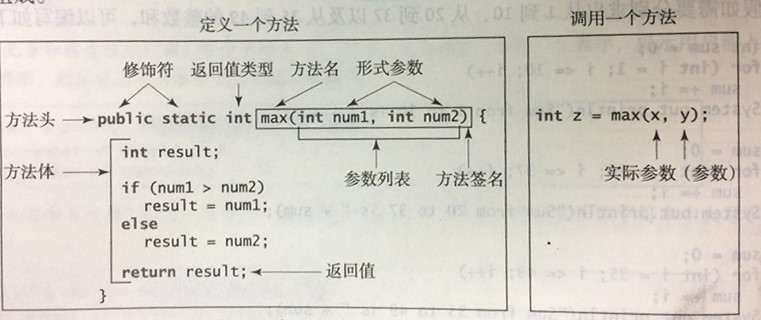
1.要点提示：方法可以用于定义可重用的代码以及组织和简化编码。

2.方法是为了完成一个操作而组合在一起的语句组。

## 6.2定义方法

1.要点提示：方法的定义由**方法名称**、**参数**、**返回值类型**以及**方法体**组成。

2.方法的定义和调用



3.带返回值的方法和void方法

带返回值的方法：如果方法有返回值，则称带返回值的方法。

Void方法：如果方法没有返回值，则称void方法。

4.形参和实参

形式参数：定义在方法头中的**变量**称为形式参数或者简称形参。

实际参数：调用方法时，给参数传递一个**值**，这个值称为实际参数或者实参。

参数是可选的，也就是说，方法可以不包含参数。

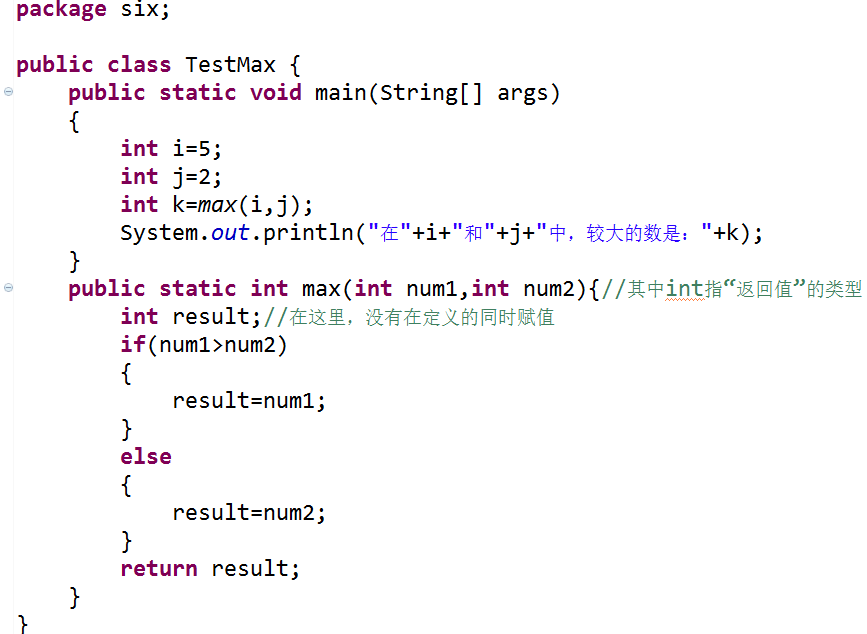
5.警告：在方法中需要对每一个参数进行独立的数据类型声明。

例如：是正确的，是错误的。

## 6.3 调用方法

1.要点提示：方法的调用是执行方法中的代码。

2.一个简单的最大值的方法调用程序。主函数调用最大值方法：



3.警告：对返回值语句来说，return语句是必须的。

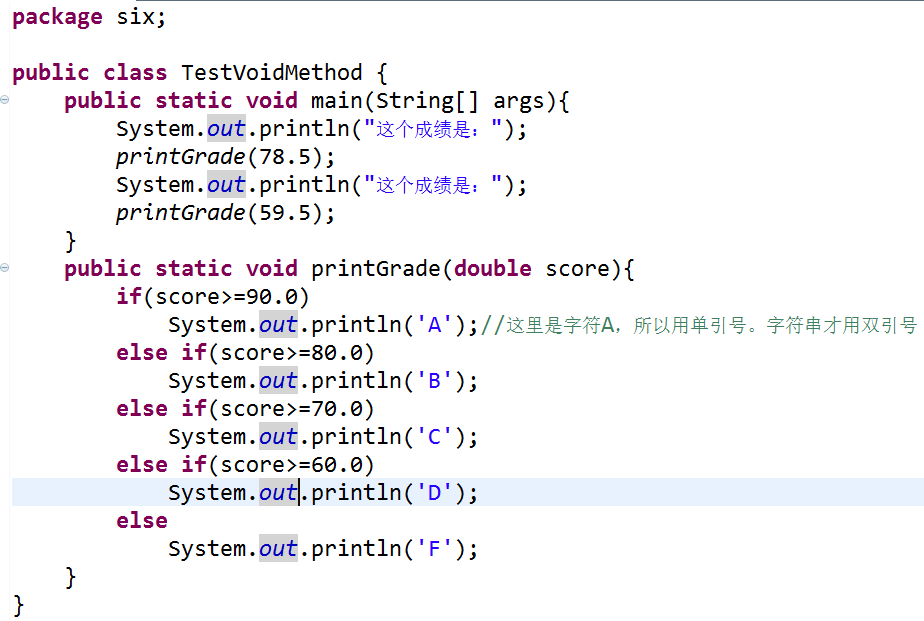
4.注意：方法能够带来代码的共享和重用。可以通过“类名(点)方法名”来调用方法，例如：



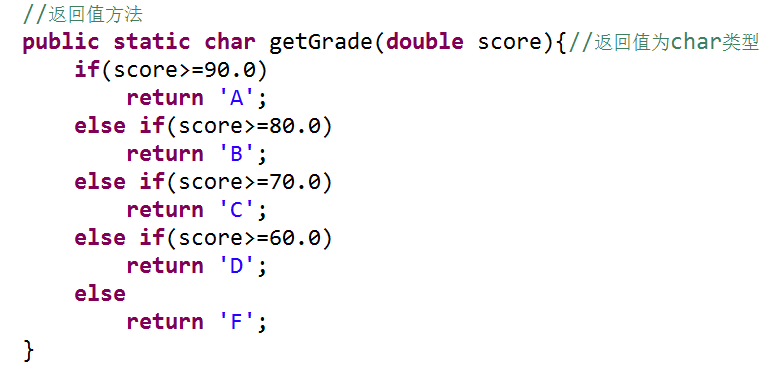
## 6.4 void方法示例

1.要点提示：void方法不返回值。

2.以下是一个根据分数评测ABCD的示例（void方法，无返回值）



3.为了区分void方法和带返回值的方法，我们重新设计printGrade方法使之返回一个值，我们称之为方法getGrade

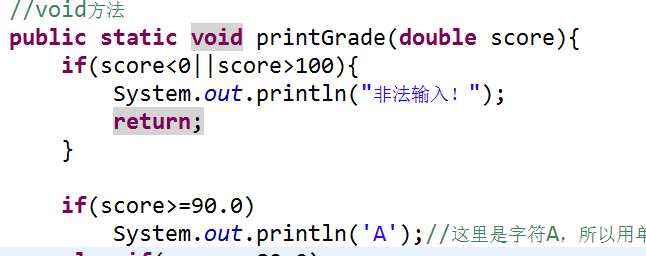


直接在主函数修改调用方法就可以直接进行调用。

4.注意：void方法不需要return语句，但它能**终止方法**并**返回到方法的调用者**。它的语法是：

这种用法很少，但是对于改变void方法中的正常流程控制是很有用的。

例如：当分数是无效值时，下列代码就用return语句结束方法。



## 6.5通过传值进行参数传递

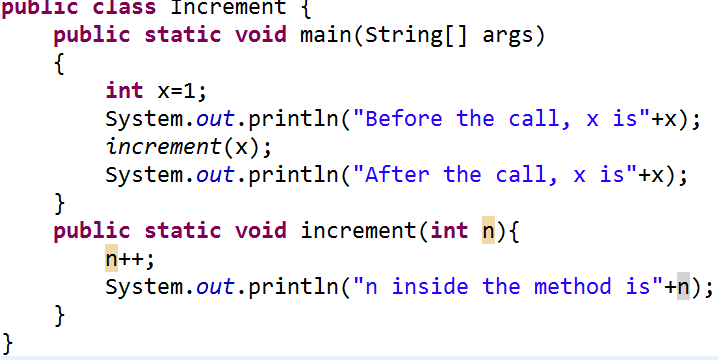
1.要点提示：调用方法的时候是通过传值的方式将实参传给形参。

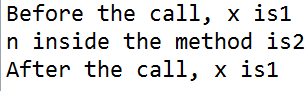
2.参数顺序匹配：调用方法时，需要提供实参，它们必须与方法签名中所对应的形参次序相同。这称作“参数顺序匹配”。

3.警告：实参必须与方法签名中定义的参数在次序和数量上匹配，在类型上兼容。

4.按值传递：当调用带参数的方法时，实参的值传递给形参，这个过程称为“按值传递”。

5. 在该方法中n自增1，而x的值则无论方法做了什么都保持不变。（什么鬼？）

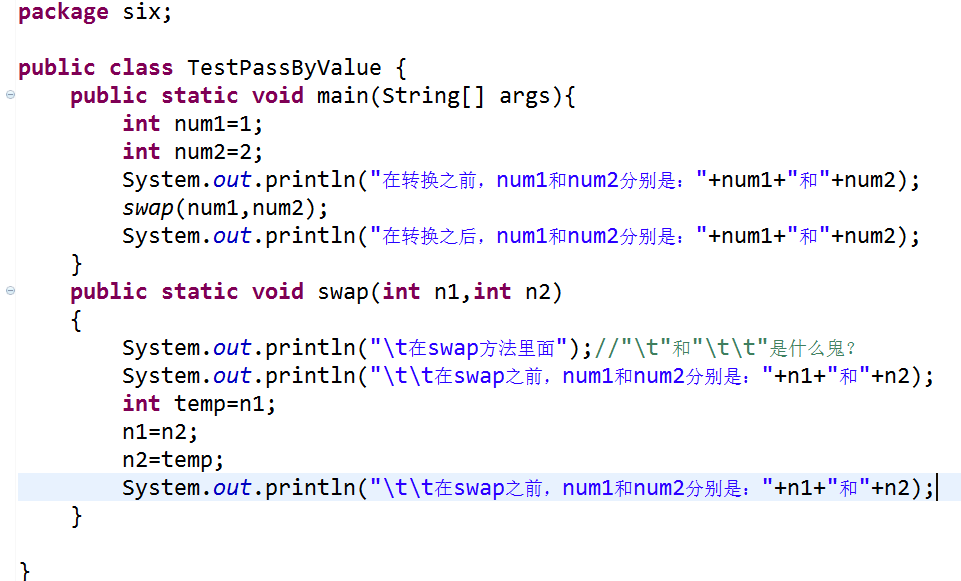


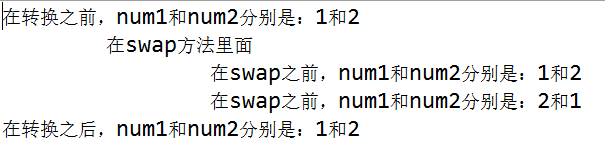


**重点原文**：如果实参是变量而不是直接量，则将该变量的值传递给形参。无论形参在方法中是否改变，该变量都不受影响。

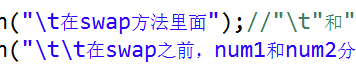
大白话：外边的是外边的，跟里面的没关系。这样？

6.以下程序创建了一个能实现交换两个变量的swap方法，调用swap方法时传递两个参数。有趣的是，调用方法后，这两个实参并未改变。（这是结果，但是为啥？）





**第一：子输出**。



**第二：关键点。**

无论参数在子方法里怎么变。一旦切换到了主方法里，参数全不变。

## 6.6 模块化代码

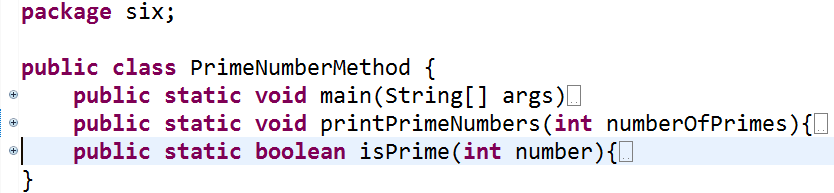
1.要点提示：模块化使得代码易于维护和调试，并且使得代码可以被重用。

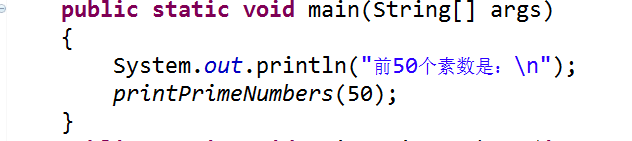
使用方法可以减少冗余代码，提高代码的复用性。方法也可以用来模块化代码，以提高程序的质量。

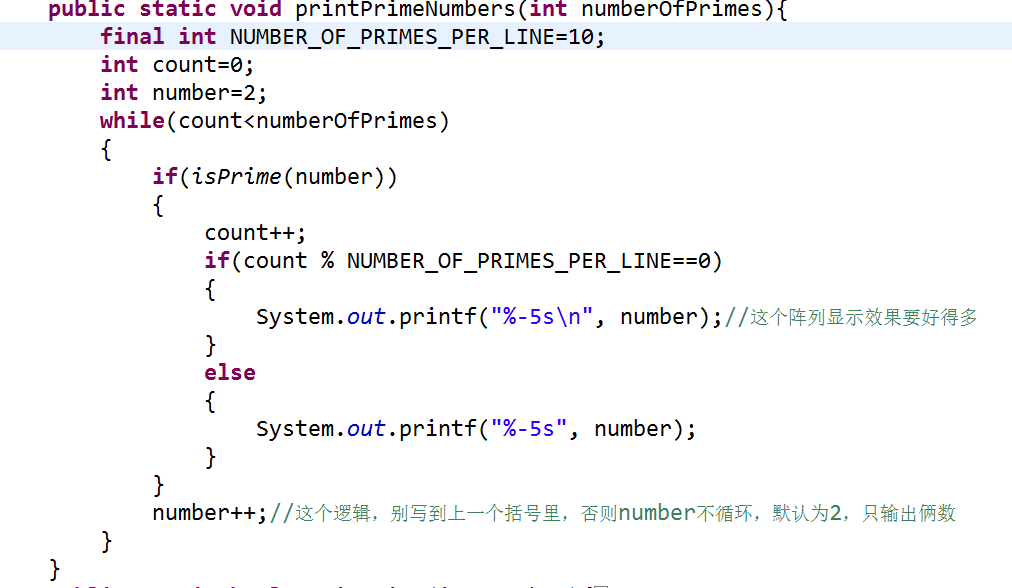
2.以下是模块化代码之后的最大公约数代码

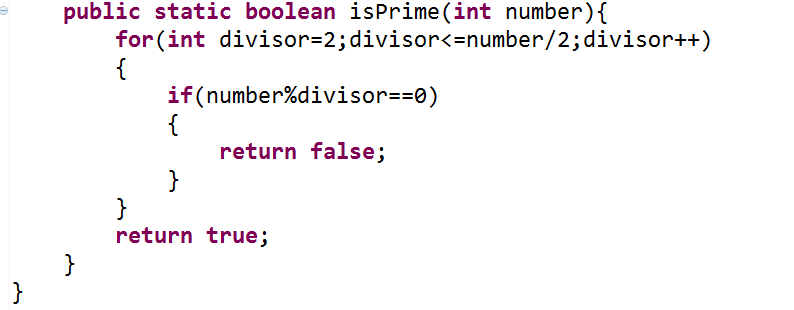


3.以下是前50个素数的算法（模块化之后的）

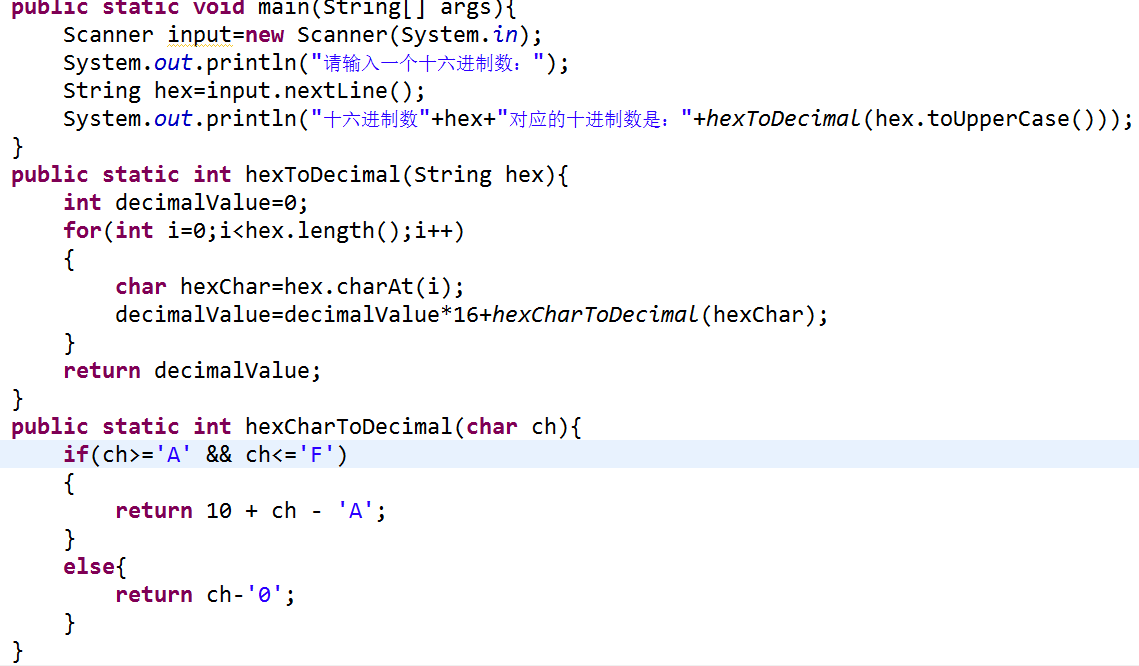








6.7 示例学习：将十六进制转换成十进制数

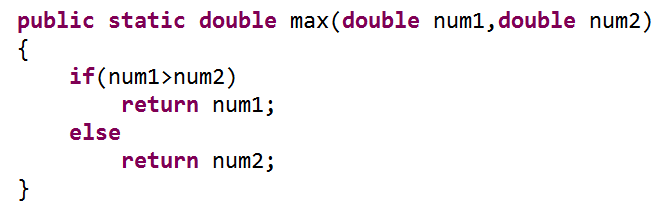


## 6.8 重载方法

1.要点提示：重载方法使得你可以使用同样的名字来定义不同的方法，只要它们的签名是不同的。

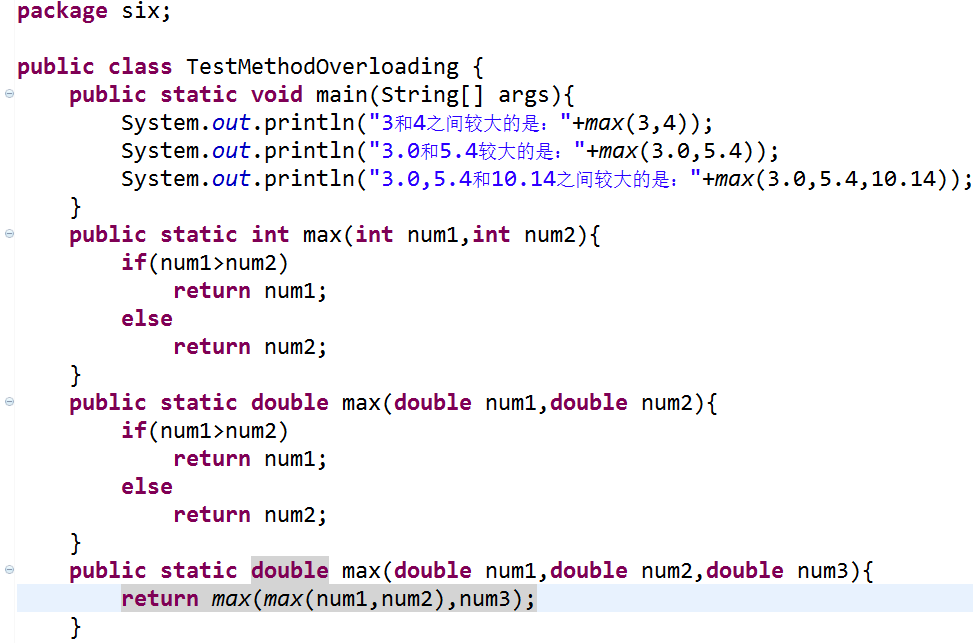
2.前面用到的max方法只能用于int型数据类型。但是，如果需要决定两个浮点数中哪个较大，该怎么办呢？

解决办法是创建另一个方法名相同但是参数不同的方法，代码如下所示：



如果调用int型参数的max方法，就将调用需要int型参数的max方法；如果调用double类型的max方法，就将调用double类型的max方法。这称为方法重载。也就是说，在一个类中有两个方法，它们具有相同的名字，但具有不同的参数列表。Java编译器根据方法签名决定使用哪个方法。

3.重载方法示例



4.提示：重载方法可以使得程序更加清晰，以及更加具有可读性。执行同样的功能但是具有不同参数类型的方法应该使用同样的名字。

5.注意：被重载的方法必须具有不同的参数列表。不能基于不同修饰符或返回值类型来重载方法。

6.注意：有时调用一个方法时，会有两个或者更多可能的匹配，但是，编译器无法判断哪个是最精确的匹配，这称为歧义调用，歧义调用会产生一个编译错误。

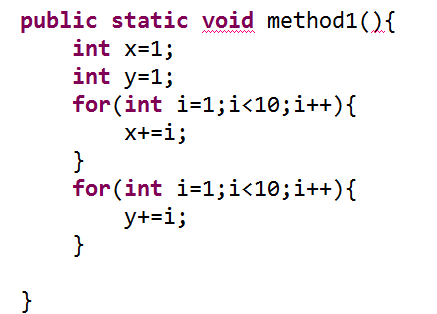
## 6.9 变量的作用域

1.要点提示：变量的作用域是指变量可以在程序中引用的范围。

2.局部变量：在方法中定义的变量称为局部变量。

3.局部变量的作用域从声明变量的地方开始，直到该变量的块结束为止。局部变量必须在使用之前进行声明和赋值。

4.参数实际上就是一个局部变量。一个方法的参数的作用域涵盖整个方法。

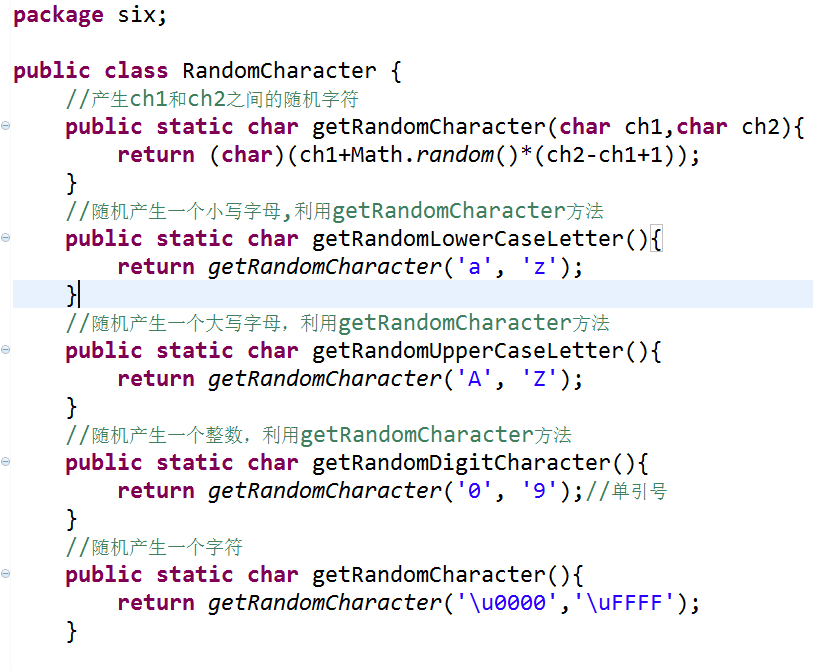


警告：不要在块内声明一个变量然后企图在块外使用它。

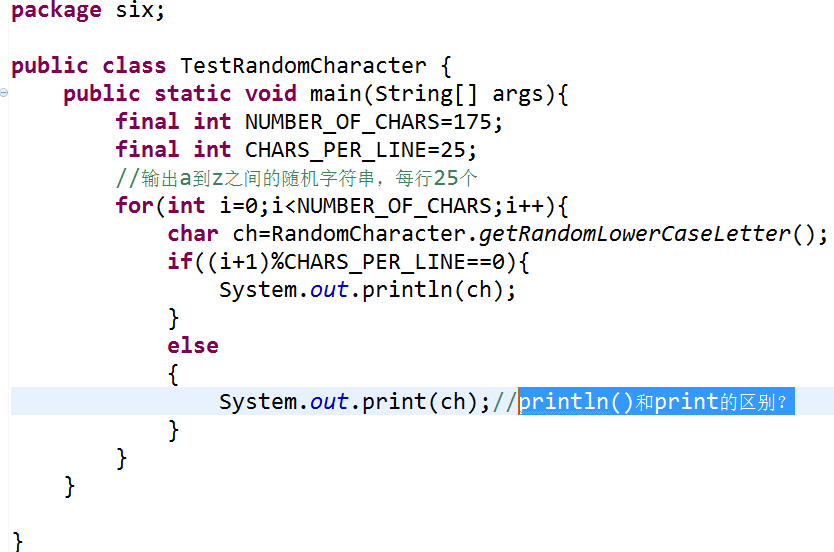
## 6.10 示例学习：产生随机字符

1.要点提示：字符使用整数来编码。产生一个随机字符就是产生一个随机整数。

2.以下程序有五个重载方法，随机获取某种特定类型的字符。



没有主函数啊？？？？？？服务于下一个类



println打印的时候自带了换行，下一次在打印的时候从下一行开始打印。  
print不换行，下次打印的时候在这次打印的东西后面直接打印。

## 6.11 方法抽象和逐步求精

1.方法抽象：通过将方法的使用和它的实现分离来实现的。

2.逐步求精：将大问题分解为子问题。子问题又分解为更小，更容易处理的问题。

### 6.11.1自顶向下的设计

由大到小，逐步细化。

### 6.11.2 自顶向下和自底向上的实现

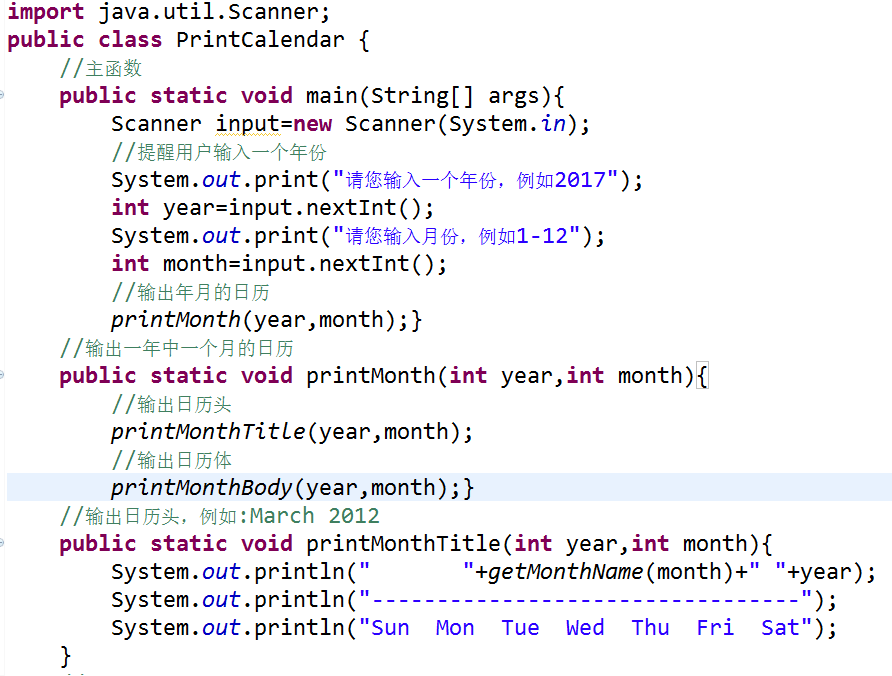
1.自顶向下：方法是自顶向下，每次实现结构图中的一个方法。

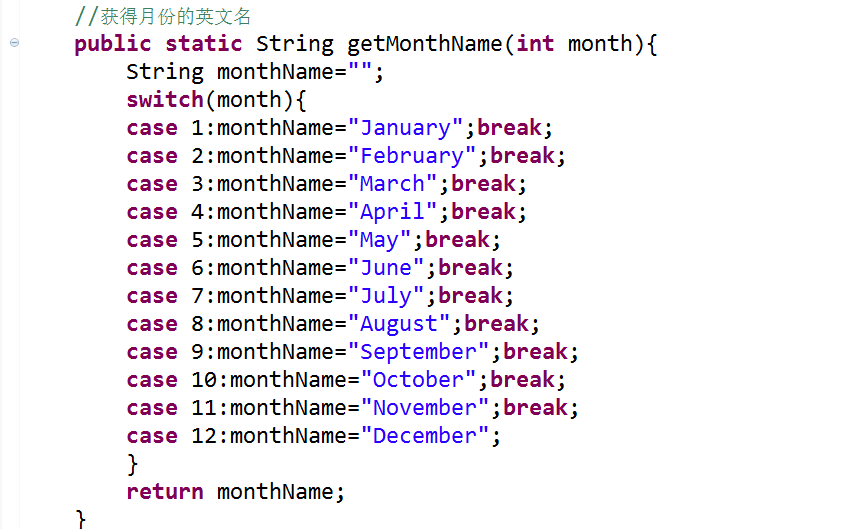
待完善方法：方法的一个简单但不够完整的版本，使用待完善方法可以快速的构建程序的框架。

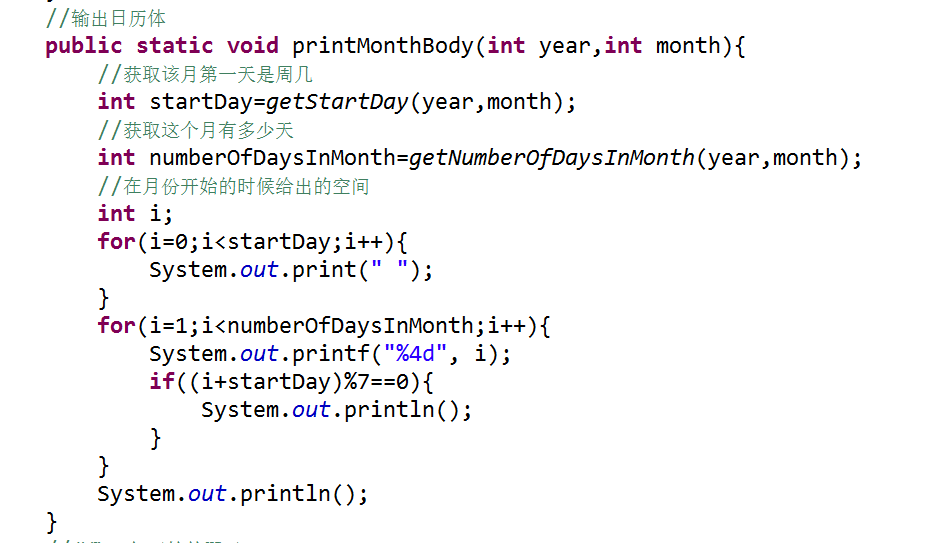
2.自底向上：从下往上每次实现结构图中的一个方法，对每个实现的方法都写一个测试程序进行测试。

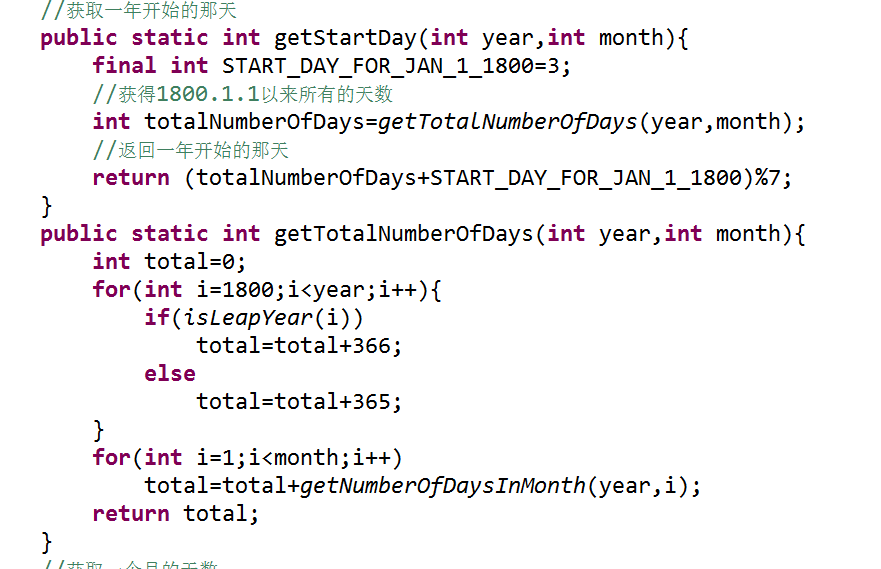
3.评价：自顶向下和自底向上都是不错的方法。它们都是逐步的实现方法，这有助于分离程序设计错误，使调试变得容易。

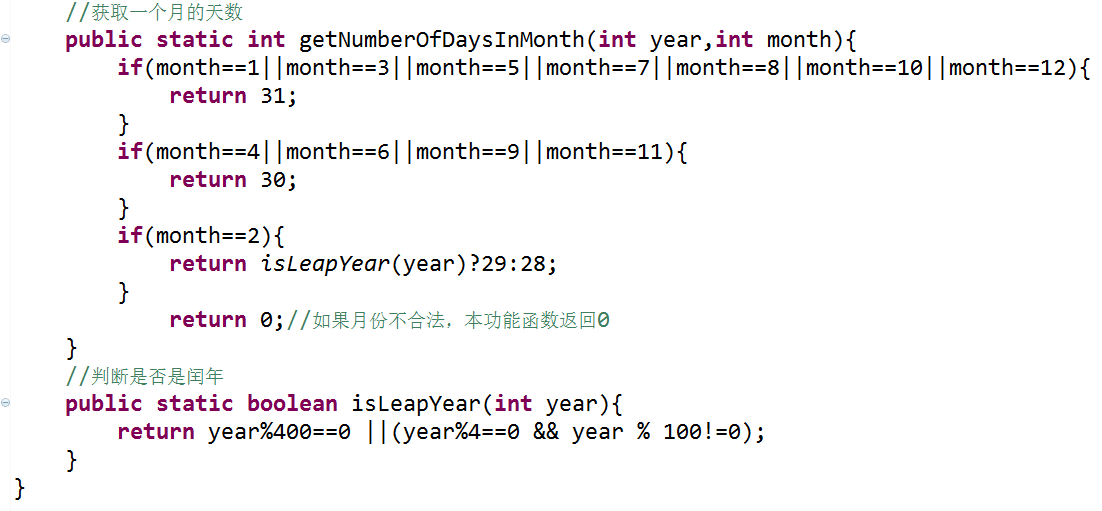
### 6.11.3 日历程序的实现细节

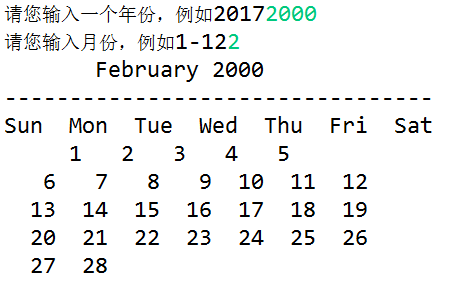












运行结果和结果排版是有错的。

### 6.11.4逐步求精的优势

1.更简单的程序。将问题分解逐步细化，便于开发，便于阅读和理解。

2.重用方法。提高一个方法的重用程度，减少冗余代码。例如：isLeap()

3.易于开发、调试和测试

4.更方便团队合作。