Recursive

一个recursive method里面包含了call自己

Recursion建立于问题——把一个任务分成多个子任务

当call一个recursive method激活时，现在进行的call暂时暂停

现在进行的call将被继续直到新的call return

同样的，如果新的method调用过程中又CALL了一个method，将转到新call的method，再转到新method，最后转回老method

当计算机遇到了一个recursive call,他肯定会暂时性的暂停method的进行，因为他需要知道recursive call的结果，在继续之前

他把所有需要的信息存储下来，这样他就可以在将来recursive call有return的时候继续运算

这个过程会中断当一个recursive call并不依赖recursion而return

一个标准的recursive method的definition

1. 一个或多个情况——这个情况对这个正在被描述的method拥有一个或多个recursive calls

这些recursive calls应该解决这个method正在解决的任务的更小版本

1. 一个或多个情况，不包括recursive calls：这个叫做base cases or stopping cases

当一个recursion被使用，这一系列的recursive call中将会call stopping case(没有recursion的method)

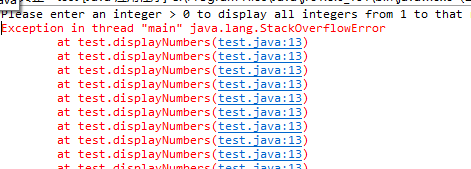
如果，每一个recursive call都会产生另外一个recursive call，那么理论上就会无限的运行下去，

这个叫做infinite recursion

在实例上，运行一段时间直到计算机run out of resources耗尽资源，然后程序不正常的中断

比如base case是n>1

Recursion每次n-1,然后n是0,理论上来说会0.-1，-2但实际上



Stack简介

为了追踪recursion，许多计算机系统会使用 stack

Stack是一种非常特殊的内存结构，有点类似于一个装了一打纸的容器，他有无限量供应的纸，一个新纸加入时被加在stack的顶部（所有之前的sheet的上面）

想移除老的sheet必须把它上面所有的sheet先remove掉（不仅要remove,实际上还要扔掉）

只要最后一张被放的sheet在stack顶上，而且这张sheet可以被拿走，那么这个stack就叫做last-in/first-out 记忆结构（LIFO）

所以为了追踪recursion，当一个method被called的时候，一个新的sheet被采用

1.Method definition被复制到sheet上，参数被作为method的参数插入

2.计算机开始运行method

3.遇到recursive，停下所有计算来进行recursive call

4.他把当前的method写在sheet of paper上，然后放在stack顶端

5.然后新的sheet of paper用来记载recursive call，复制了method definition，但是参数不同

6.当这个copy 遇到recursive call，同样存储信息，加入新的sheet of paper

这一过程直到没有recursive call的base class

然后计算机回到stack顶端，一层一层解决未完成的,正在等待的运算，解决一层，丢掉一层

Stack还有一个配套产品

仅存放一张sheet of paper的内容的叫做stack frame/activation record

Stack frame并不能容纳一份完整的method definition的copy，它容纳的只是一个copy的reference

Stack OverFlow

Stack的size实际上总会有些限制

如果有一个长的连锁反应当一个Method call自己的时候，（疯狂recursive），这时候就会有很多被暂停的计算积压在stack上

如果有太多，stack就会尝试增长大小来超越limit，这时候就会导致一个error情况叫做stack overflow

一个经常导致stack overflow的就是inifinite recursion

Doc3

RECURSION VS ITERATIVE

Recursion并不是绝对需要的

任何可以被recursion解决的任务都可以用nonrecursive 方式解决

一个nonrecursive 版本的method叫做iterative version

一个iteratively written的method将会使用loop来取代recursion

一个recursively written的method更简单，但是跑得慢，用的存储更多，与iterative version相比较

Recursive method that return a value

Recursion 并不限制一定要是void

一个recursive method 可以return任意类型的value

一个成功的要return value的模板：

1. 一个或多个情况return的值基于recursive call
2. Recursive call得参数应该更小
3. 一个或多个情况，return的值不基于任何recursive call(base/stoppling)

**public** **static** **int** sumNumbers(**int** n)//返回的是int

{

**if**(n <= 0)

{

System.*out*.println("You must enter an integer larger than 0." +

" Method will return \"0\" as an indication of no calculations.");

**return** 0;//return一个0,base

}

**int** total = 0;

// Simple Case (Basic Step)

**if**(n == 1 )

{

total += 1;

**return** total;//不依赖recursive

}

**else**

{

// Call myself (that same method) to calculate the total for up to n-1 then add n

total = n + *sumNumbers*(n-1);

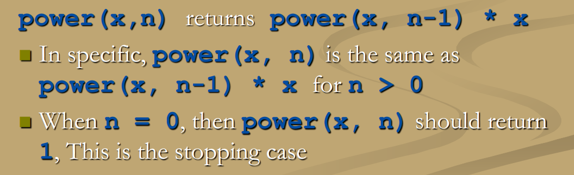
**return** total;//recursive，注意return的值一定要和recursive有关

}

}

Thinking Recursively

如果一个问题需要recursion，比起专注于stack和暂停的计算，我们更需要专注于，进行了几次循环



1. 绝不能有infinite recursion,所有的recursive连锁循环最终一定要到stopping case
2. 每个stopping case都会return正确的值
3. 对于有recursion的case，只要每个循环都能return正确的值，最终就能return正确的值
4. 这些性质叫做mathematical induction

Binary Search

Binary search使用recursive method来搜索一个sorted（有序数的，排好号的）array来找到一个特殊值

Array必须是sorted array .

如果一个值被找到，会return一个index.没找到,会return一个-1

注意：每一次使用recursive method都会减少一半的·搜索空间（半分法）

这个算法解决任务的时候从array的中间或array segement开始

如果这个值被在这个index（中间值）找到了，return这个index结束搜索

如果在想要寻找的值比array中间的值要小，那么忽略下半部分。要大于，忽略上把部分

然后对前半部分重复以上步骤

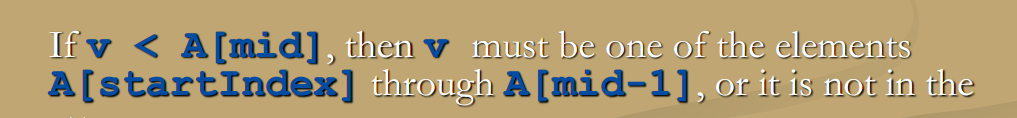
直到循环完也没找到，返回-1

要注意中间数的加一减一

Doc6

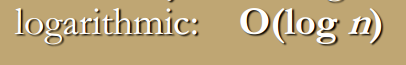
不会遇到无限循环

每一个recursive call，要么startIndex值增加，要么endIndex值变小，最后startIndex总会大于endIndex





Binary search效率远比一个一个查要高

Binary search中效率最低的一类：对数法

如果有需要 search可以从recursive版本转到iterative 版本

Doc7