

知识点列表

编号	名称	描述	级别
1	选择排序	Java 数组排序，企业招聘程序员笔试时经常考到，须熟练掌握	**
2	冒泡排序	Java 数组排序，企业招聘程序员笔试时经常考到，须熟练掌握	**
3	插入排序	Java 数组排序，企业招聘程序员笔试时经常考到，须熟练掌握	**
4	Java 系统排序	掌握 Java API 中排序的方法	***
5	方法的递归调用	理解递归调用的原理	**

注： **"理解级别 ***"掌握级别 ****"应用级别

目录

1. Java 数组排序.....	3
1.1. 选择排序 **.....	3
1.2. 冒泡排序 **.....	4
1.3. 插入排序 **.....	7
2. java 系统排序 ***.....	11
3. 方法的递归调用 **.....	15

1. Java 数组排序

关于今天的学习内容，需要提醒大家如下几点：

- ✓ 学习目的：不是为了在工作中使用（这三个算法性能不高），而是为了练习 for 循环和数组
- ✓ 在工作中 Java API 提供了现成的优化的排序方法，效率很高，以后工作中直接使用即可
- ✓ 选择排序、冒泡排序、插入排序等排序算法在公司笔试题中会经常出现，所以是很有必要掌握的

1.1. 选择排序 **

选择排序原理：

- ✓ 将数组中每个元素与第一个元素比较，如果这个元素小于第一个元素，则交换这两个元素
- ✓ 循环第 1 条规则，找出最小元素，放于第 1 个位置
- ✓ 经过 n-1 轮比较完成排序

简单而言，每轮都找到最小的放到前面。举例：{8, 2, 3, 7, 1}的排序过程如下所示：

数组： ary={8, 2, 3, 7, 1}

第 1 轮： ary={1 | 8, 3, 7, 2}

第 2 轮： ary={1, 2 | 8, 7, 3}

第 3 轮： ary={1, 2, 3 | 8, 7}

第 4 轮： ary={1, 2, 3, 7 | 8}

第 5 轮： ary={1, 2, 3, 7 | 8}

过程分析：

- ✓ i 代表第一个数据的位置
- ✓ j 代码后部每一个数据的位置

ary	i	j	ary[i]	ary[j]	ary[i]>ary[j]	[i]交换[j]
第 1 轮						
{8 2,3,7,1}	[0]	[1]	8	2	true	8<->2
{2 8,3,7,1}	[0]	[2]	2	3	false	--
{2 8,3,7,1}	[0]	[3]	2	7	false	--
{2 8,3,7,1}	[0]	[4]	2	1	true	2<->1
{1,8 3,7,2}						
第 2 轮						
{1,8 3,7,2}	[1]	[2]	8	3	true	8<->3
{1,3 8,7,2}	[1]	[3]	3	7	false	--
{1,3 8,7,2}	[1]	[4]	3	2	true	3<->2
{1,2,8 7,3}						
第 3 轮						
{1,2,8 7,3}	[2]	[3]	8	7	true	8<->7
{1,2,7 8,3}	[2]	[4]	7	3	true	7<->3
{1,2,3,8 7}						

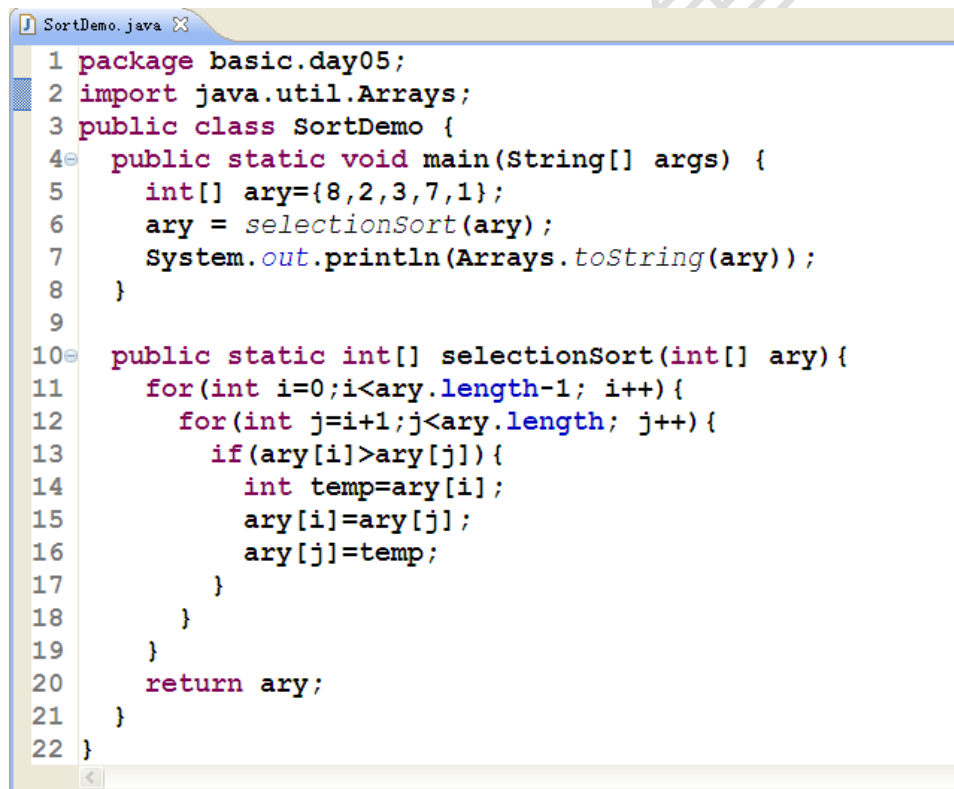
第 4 轮						
{1,2,3,8 7}	[3]	[4]	8	7	true	8<->7
{1,2,3,7 8}						
第 5 轮						
{1,2,3,7,8 }	[4]	--	8	--	--	--

备注：

- ✓ i 的范围是： 0 ~ <ary.length - 1
- ✓ j 的范围是： i+1 ~ <ary.length
- ✓ 交换步骤 (ary[i]<->ary[j]) 的代码如下：

```
if(ary[i]>ary[j]){
    int temp = ary[i];
    ary[i]=ary[j];
    ary[j]=temp;
}
```

参考代码如下所示：



```
SortDemo.java X
1 package basic.day05;
2 import java.util.Arrays;
3 public class SortDemo {
4     public static void main(String[] args) {
5         int[] ary={8,2,3,7,1};
6         ary = selectionSort(ary);
7         System.out.println(Arrays.toString(ary));
8     }
9
10    public static int[] selectionSort(int[] ary){
11        for(int i=0;i<ary.length-1; i++){
12            for(int j=i+1;j<ary.length; j++){
13                if(ary[i]>ary[j]){
14                    int temp=ary[i];
15                    ary[i]=ary[j];
16                    ary[j]=temp;
17                }
18            }
19        }
20        return ary;
21    }
22 }
```

1.2. 冒泡排序 **

冒泡排序原理：

- ✓ 比较相邻的元素，将小的放到前面。

冒泡排序举例：{8, 2, 3, 7, 1}的排序过程如下所示：

```
ary={8,2,3,7,1}
ary={2,8,3,7,1}
ary={2,3,8,7,1}
ary={2,3,7,8,1}
ary={2,3,7,1|8}
ary={2,3,7,1|8}
ary={2,3,7,1|8}
ary={2,3,1|7,8}
ary={2,3,1|7,8}
ary={2,1|3,7,8}
ary={1,2,3,7,8}
```

过程分析:

- ✓ i 代表次数
- ✓ j 代表比较位置

ary	i	j	j+1	ary[j]	ary[j+1]	ary[j]>ary[j+1]	[j]交换[j+1]
{8,2,3,7,1}	0	[0]	[1]	8	2	true	8<->2
{2,8,3,7,1}	0	[1]	[2]	8	3	true	8<->3
{2,3,8,7,1}	0	[2]	[3]	8	7	true	8<->7
{2,3,7,8,1}	0	[3]	[4]	8	1	true	8<->1
{2,3,7,1 8}	1	[0]	[1]	2	3	false	--
{2,3,7,1 8}	1	[1]	[2]	3	7	false	--
{2,3,7,1 8}	1	[2]	[3]	7	1	true	7<->1
{2,3,1 7,8}	2	[0]	[1]	2	3	false	--
{2,3,1 7,8}	2	[1]	[2]	3	1	true	3<->1
{2,1 3,7,8}	3	[0]	[1]	2	1	true	2<->1
{1,2,3,7,8}							

备注：

- ✓ i 的取值范围是： i = 0 ~ <ary.length-1
- ✓ j 的取值范围是： j = 0 ~ <ary.length - i - 1
- ✓ 交换步骤伪代码如下：

```
if(j>j+1){
    [j]<->[j+1]
}
```

参考代码如下所示：

```
SortDemo.java X
1 package basic.day05;
2 import java.util.Arrays;
3 public class SortDemo {
4     public static void main(String[] args) {
5         int[] ary={8,2,3,7,1};
6         ary = bubbleSort(ary);
7         System.out.println(Arrays.toString(ary));
8     }
9
10    public static int[] bubbleSort(int[] ary){
11        for(int i=0;i<ary.length-1; i++){
12            for(int j=0;j<ary.length-i-1; j++){
13                if(ary[j]>ary[j+1]){
14                    int temp=ary[j];
15                    ary[j]=ary[j+1];
16                    ary[j+1]=temp;
17                }
18            }
19        }
20        return ary;
21    }
22 }
```

如果对排序的执行过程还有疑惑，可以使用**输出语句**观察执行结果，在写程序的过程中，程序员常常使用输出语句进行调试，如图所示：

```

SortDemo.java X
9
10 public static int[] bubbleSort(int[] ary){
11     for(int i=0;i<ary.length-1; i++){
12         for(int j=0;j<ary.length-i-1; j++){
13             //状态跟踪语句, 状态=数据=当前变量的值
14             //Debug: 将跟踪的结果和理想的结果比较, 找错误
15             System.out.print(Arrays.toString(ary)+"\t");
16             System.out.print(i+"\t");
17             System.out.print(j+"\t");
18             System.out.print(ary[j]+" \t");
19             System.out.print(ary[j+1]+" \t");
20             System.out.print((ary[j]>ary[j+1])+"\t");
21             if(ary[j]>ary[j+1]){
22                 System.out.print(ary[j]+"<->" +ary[j+1]);
23                 int temp=ary[j];
24                 ary[j]=ary[j+1];
25                 ary[j+1]=temp;
26             }
27             System.out.println();
28         }
29     }
30     return ary;
31 }
32 }

```

输出结果如下：

```

Console X
<terminated> SortDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 8, 2011 2:47:29 PM)
[8, 2, 3, 7, 1] 0      0      8      2      true      8<->2
[2, 8, 3, 7, 1] 0      1      8      3      true      8<->3
[2, 3, 8, 7, 1] 0      2      8      7      true      8<->7
[2, 3, 7, 8, 1] 0      3      8      1      true      8<->1
[2, 3, 7, 1, 8] 1      0      2      3      false
[2, 3, 7, 1, 8] 1      1      3      7      false
[2, 3, 7, 1, 8] 1      2      7      1      true      7<->1
[2, 3, 1, 7, 8] 2      0      2      3      false
[2, 3, 1, 7, 8] 2      1      3      1      true      3<->1
[2, 1, 3, 7, 8] 3      0      2      1      true      2<->1
[1, 2, 3, 7, 8]

```

1.3. 插入排序 **

在开始学习插入排序之前，让我们先对 for 循环做个补充：如下是关于 for 循环结束时，i 的取值的

演示代码，如图所示：

```
package basic.day05;

public class ForDemo {
    public static void main(String[] args) {
        int i;
        for(i=0; i<5; i++){//不满足i<5时候结束循环，结束时候i=5
            System.out.println(i);
        }
        System.out.println("结束时候："+i);//5
        for(i=5; i>=0; i--){//i>=0返回false结束，结束时候i=-1
            System.out.println(i);
        }
        System.out.println("结束时候："+i);//-1
    }
}
```

如图所示，当 i 的值不满足 for 循环条件，循环结束。

插入排序原理：

- ✓ 将数组分为两部分，将后部分的第一个逐一与前部分每一个元素比较，在合理位置插入
- ✓ 插入排序算法效率要高于选择排序和冒泡排序

插入排序举例：{8, 2, 3, 7, 1}的排序过程如下所示：

- | | |
|--|-------------|
| 第 1 步，假设第一个元素是已排序的 | {8 2,3,7,1} |
| 第 2 步，用 2 和 " " 之前的所有元素比较，并插入 | {8 2,3,7,1} |
| 取出 2 (temp=2) | |
| temp 和 8 比，比 8 小，将 2 的位置赋值为大数 (ary[1]=8) | {8 8,3,7,1} |
| 因为已到边界，直接赋值 (ary[0]=2) | {2,8 3,7,1} |
| 2 和 8 排序完成 | |
| 第 3 步，用 3 和 " " 之前的所有元素比较，并插入 | {2,8 3,7,1} |
| 取出 3 (temp=3) | |
| temp 和 8 比，比 8 小，3 的位置赋值给大数 (ary[2]=8) | {2,8 8,7,1} |
| temp 和 2 比，比 2 大，插入 2 后面 (ary[1]=3) | {2,3,8 7,1} |
| 3、2、8 排序完成 | |
| 第 4 步，用 7 和 " " 之前的所以元素比较，并插入 | {2,3,8 7,1} |
| 取出 7 (temp=7) | |
| temp 和 8 比，比 8 小，7 的位置赋值给大数 (ary[3]=8) | {2,3,8 8,1} |

temp 和 3 比, 比 3 大, 插入 3 后面 (ary[2]=7) {2,3,7,8|1}
 7、2、3、8 排序完成
 第 5 步, 用 1 和 "|" 之前的所以元素比较, 并插入 {2,3,7,8|1}
 取出 1 (temp=1)
 temp 和 8 比, 比 8 小, 1 的位置赋值给大数 8 {2,3,7,8|8}
 temp 和 7 比, 比 7 小, 8 的位置赋值给大数 7 {2,3,7,7|8}
 temp 和 3 比, 比 3 小, 7 的位置赋值给大数 3 {2,3,3,7|8}
 temp 和 2 比, 比 2 小, 3 的位置赋值给大数 2 {2,2,3,7|8}
 到边界, 赋值 (ary[0]=1) {1,2,3,7,8|}
 1、2、3、7、8 排序完成

过程分析:

- ✓ temp 代表取出待插入的元素
- ✓ i 代表后组待插入元素的位置
- ✓ j 代表前组每个元素的位置

ary	i	temp	j	ary[j]	temp<ary[j]	[j]移动[j+1]	t 插入[j+1]
{8 2,3,7,1}	[1]	2	[0]	8	true	8->[j+1]	--
{8 8,3,7,1}	[1]	2	[-1]	--	--	--	2->[j+1]
{2,8 3,7,1}	[2]	3	[1]	8	true	8->[j+1]	--
{2,8 8,7,1}	[2]	3	[0]	2	false	--	3->[j+1]
{2,3,8 7,1}	[3]	7	[2]	8	true	8->[j+1]	--
{2,3,8 8,1}	[3]	7	[1]	3	false	--	7->[j+1]
{2,3,7,8 1}	[4]	1	[3]	8	true	8->[j+1]	--
{2,3,7,8 8}	[4]	1	[2]	7	true	7->[j+1]	--
{2,3,7,7 8}	[4]	1	[1]	3	true	3->[j+1]	--
{2,3,3,7 8}	[4]	1	[0]	2	true	2->[j+1]	--
{2,2,3,7 8}	[4]	1	[-1]	--	--	--	1->[j+1]
{1,2,3,7,8 }	[5]	--	--	--	--	--	--

备注:

- ✓ i 的取值范围是: i=1 ~ <ary.length i++
- ✓ j 的取值范围是: j= i-1 ~ >=0, j-- j--
- ✓ 伪代码如下:

```
temp = [i];
if(temp<[j]){
    [j]->[j+1]    //移动
}else{
    break j;
}
temp->[j+1];      //插入
```

参考代码如下图所示:

SortDemo01.java

```
SortDemo.java X
1 package basic.day05;
2 import java.util.Arrays;
3 public class SortDemo {
4     public static void main(String[] args) {
5         int[] ary={8,2,3,7,1};
6         ary = insertSort2(ary);
7         System.out.println(Arrays.toString(ary));
8     }
9     /** 插入式排序*/
10    public static int[] insertSort2(int[] ary){
11        int i,j,t;
12        for(i=1; i<ary.length; i++){
13            t = ary[i];
14            //利用循环查找插入位置:到头j=-1
15            for(j=i-1; j>=0; j--){
16                if(t<ary[j]){
17                    ary[j+1]=ary[j];//[j]->[j+1]向后移动
18                }else{//t不小于ary[j]
19                    break;//找到插入位置: t不小于ary[j]
20                }
21            }
22            ary[j+1]=t;//插入t->[j+1]
23        }
24        return ary;
25    }
}
```

SortDemo02.java (经典写法)

```
SortDemo.java X
1 package basic.day05;
2 import java.util.Arrays;
3 public class SortDemo {
4     public static void main(String[] args) {
5         int[] ary={8,2,3,7,1};
6         ary = insertSort(ary);
7         System.out.println(Arrays.toString(ary));
8     }
9
10    public static int[] insertSort(int[] ary){
11        //ary = 8,3,5,1,7
12        int i,j,t;
13        for(i=1; i<ary.length; i++){
14            t = ary[i];
15            //利用循环查找插入位置:到头j=-1
16            for(j=i-1; j>=0 && t<ary[j]; j--){
17                ary[j+1]=ary[j]; //[j]->[j+1] 向后移动
18            }
19            ary[j+1]=t; //插入t->[j+1]
20        }
21        return ary;
22    }
23 }
```

2. java 系统排序 ***

JDK 提供的排序方法 `Arrays.sort(ary)` 的效率要比我们之前写的选择排序、冒泡排序等效率高。

在做比较之前，先介绍一个方法 `System.currentTimeMillis()`，通过该方法可以得到格林威治时间自 1970 年 1 月 1 日凌晨 00:00:00 到当前系统时间的**毫秒数**（返回值为 long 类型）。如图所示：

```
Test.java
1 package basic;
2 /** Java提供的数组复制方法*/
3 public class Test {
4     public static void main(String[] args) {
5         long time = System.currentTimeMillis();
6         System.out.println(time);
7     }
8 }
```

```
Console
<terminated> Test [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 9, 2011 3:18:08
1318144688531
```

在这里有一个疑问，long 类型的范围表示够不够大？我们做个测试：

第 1 步，我们可以通过这个方法得到当前的年份，如图所示：

```
Test.java
1 package basic;
2 /** Java提供的数组复制方法*/
3 public class Test {
4     public static void main(String[] args) {
5         long now = System.currentTimeMillis();
6         long year = now/1000/60/60/24/365+1970;
7         System.out.println(year); //当前系统时间是2011年
8     }
9 }
10
```

```
Console
<terminated> Test [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 9, 2011 3:48:27 PM
2011
```

第 2 步，long 类型能表示的最大年份，如图所示：

```
Test.java
1 package basic;
2 /** Java提供的数组复制方法*/
3 public class Test {
4     public static void main(String[] args) {
5         long max = 0x7fffffffffffffffL;
6         long maxYears = max/1000/60/60/24/365+1970;
7         System.out.println(maxYears); //当前系统时间是2011年
8     }
9 }
```

Console

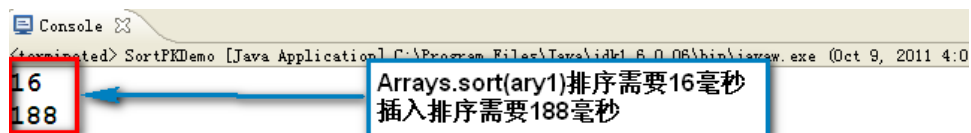
292473178

long类型能表示上亿年，足够用了

接下来我们回归主题，比较 Arrays.sort(ary)和插入排序的效率，如图所示：

```
SortPKDemo.java
1 package basic.day05;
2 import java.util.Arrays;
3
4 public class SortPKDemo {
5     public static void main(String[] args) {
6         int[] ary = new int[20000];
7         Random r = new Random();
8         for(int i=0; i<ary.length; i++){
9             //随机生成一个整数赋值给数组元素
10            ary[i] = r.nextInt();
11        }
12        //拷贝两个数组副本
13        int[] ary1 = Arrays.copyOf(ary, ary.length);
14        int[] ary2 = Arrays.copyOf(ary, ary.length);
15
16        //计算Arrays.sort()排序花费的时间
17        long start = System.currentTimeMillis();
18        Arrays.sort(ary1);
19        long end = System.currentTimeMillis();
20        System.out.println(end-start);
21
22        //计算插入排序算法排序花费的时间
23        start = System.currentTimeMillis();
24        InsertSort01.insertSort(ary2);
25        end = System.currentTimeMillis();
26        System.out.println(end-start);
27    }
28 }
```

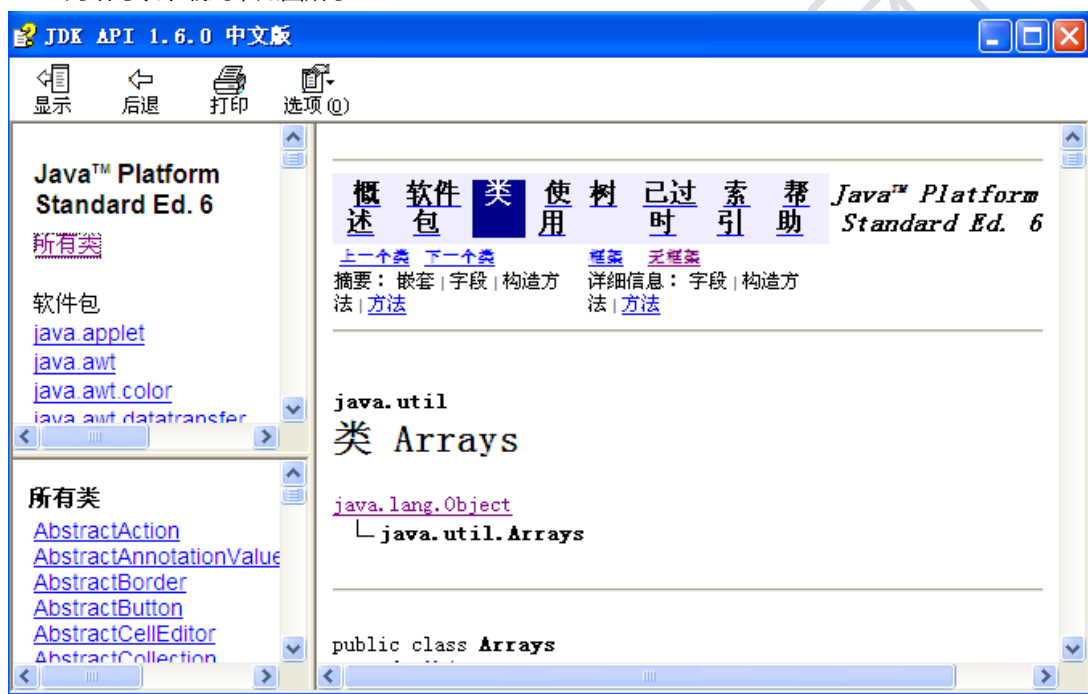
运行结果：



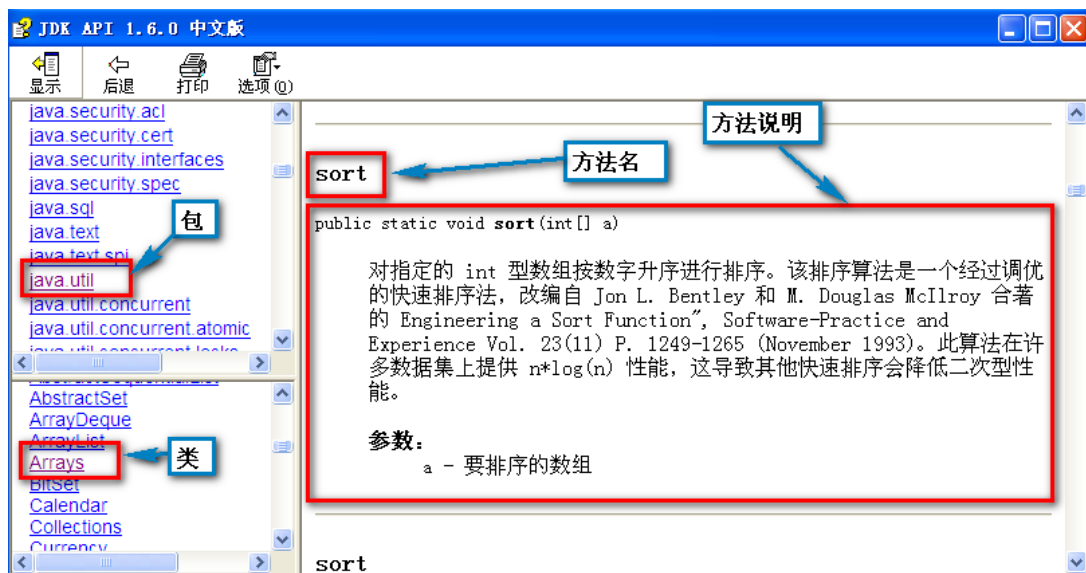
注意：

- ✓ 运行结果为 0 表示不到 1 毫秒内就完成了排序
- ✓ 不要试图在控制台输出数组元素，控制台打印输出不全

JDK 提供的方法（比如 `Arrays.sort(ary)`）在官方提供的 API 帮助文档中可以查询到，在 Oracle 的官方网站可以下载到，如图所示：



我们可以通过该文档查询到 JDK 提供的类及方法如何使用，比如 `Arrays.sort()` 方法如图所示：

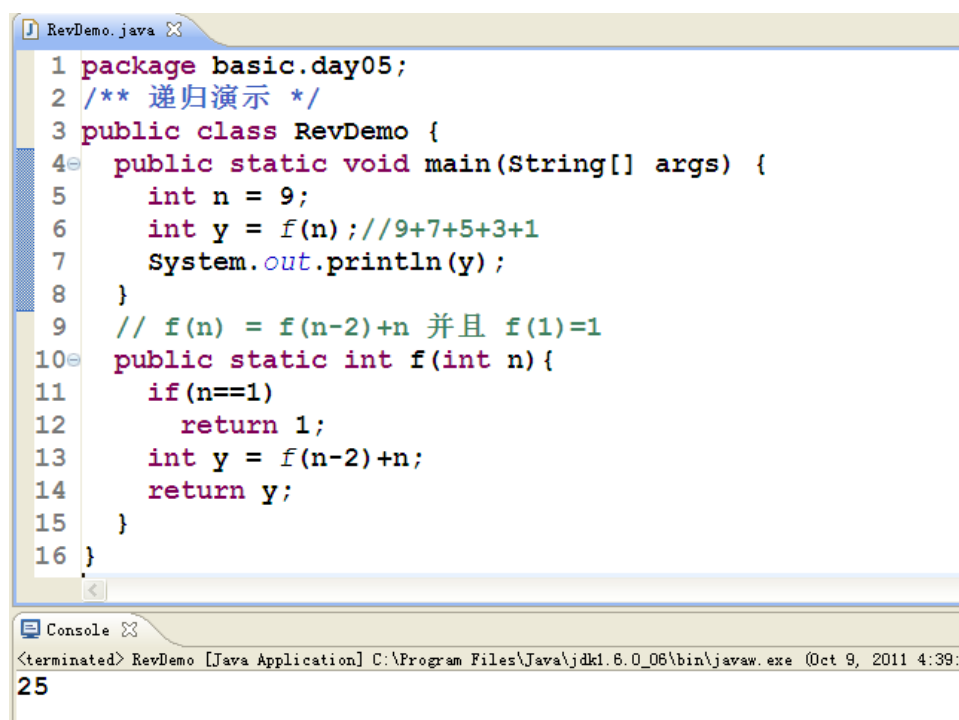


3. 方法的递归调用 **

方法的递归调用类似数学归纳法。

用公式表示： $y = f(n) = 1 + 3 + 5 + 7 + \dots + (n-2) + n = f(n-2) + n$ ，并且 $f(1) = 1$ 。诸如 $f(n) = f(n-2) + n$ 这种函数调用函数本身的形式被称为**递归调用**。

如下案例利用递归调用计算出 $9 + 7 + 5 + 3 + 1$ 的值，参考代码如下所示：



```
1 package basic.day05;
2 /** 递归演示 */
3 public class RevDemo {
4     public static void main(String[] args) {
5         int n = 9;
6         int y = f(n); // 9+7+5+3+1
7         System.out.println(y);
8     }
9     // f(n) = f(n-2)+n 并且 f(1)=1
10    public static int f(int n) {
11        if (n==1)
12            return 1;
13        int y = f(n-2)+n;
14        return y;
15    }
16 }
```

Console

<terminated> RevDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 9, 2011 4:39:25)

25

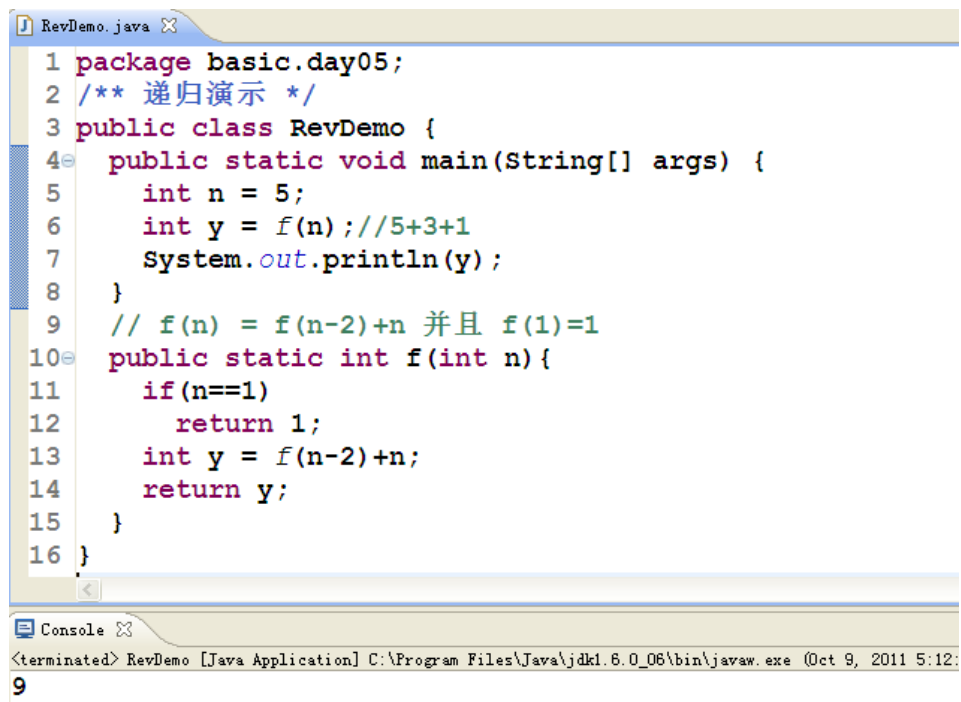
在学习递归实现原理之前，我们先了解一下**栈内存**。

栈内存是计算机中的一种数据存储方式，是 Java 进程启动时候在内存中开辟的存储空间。

- ✓ **栈内存**的利用方式遵循 LIFO(后进先出)原则
- ✓ Java 所有局部变量都在栈中分配(压入)，方法的参数也是局部变量，局部变量在离开作用域时候回收，就是从栈中弹出(删除)。
- ✓ Java 中所有的局部变量都是在栈内存中分配的（包括方法中声明的变量、方法的参数）。

Java 方法调用使用栈实现，递归调用就是栈实现的。递归时候要按照递归深度分配全部临时变量，栈开销很大，性能不好，要注意不要超过栈的大小，并且一定要给出结束条件，否则会造成栈溢出错误。

接下来我们通过递归计算 $5+3+1$ 演示递归调用过程，如下图所示：



```

1 package basic.day05;
2 /** 递归演示 */
3 public class RevDemo {
4     public static void main(String[] args) {
5         int n = 5;
6         int y = f(n); // 5+3+1
7         System.out.println(y);
8     }
9     // f(n) = f(n-2)+n 并且 f(1)=1
10    public static int f(int n){
11        if(n==1)
12            return 1;
13        int y = f(n-2)+n;
14        return y;
15    }
16 }

```

Console

```

<terminated> RevDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 9, 2011 5:12:
9

```

程序执行关键步骤如下图所示：

第 1 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

stack 栈

栈内存中为局部变量n开辟一块32位的空间

int n = 5

第2步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

赋值运算，先计算“=”右边的表达式

stack 栈

int n = 5

第 3 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

实参

形参

调用方法f(int n)，如果有参数，则在栈内存中为参数(形参)开辟空间，并将实参赋值给形参

stack 栈

int n = 5

int n = 5

第 4 步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

stack 栈

int n = 5

int n = 5

开始执行方法体内部代码

第 5 步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

stack 栈

int n = 5

int n = 5

n=5, if(n==1)条件不成立
执行int y = f(n-2)+n
先执行"="右边表达式

第 5 步补充说明

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

stack 栈

int n = 5

int n = 5

f(n-2)+n 执行步骤:

1. 先计算参数表达式的值 $n-2=3$
2. 调用 f(3)
3. 当 f(3) 获得返回值后才执行 "+" 运算 *

第 6 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

stack 栈

int n = 3

int n = 5

int n = 5

1. 栈内存中分配临时变量 $\text{int } n = 3$
2. 为方法 f(int n) 传入参数 $n = 3$
3. 调用 f(3)
4. f(5) 挂起

第 7 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

n=3, if(n==1)条件不满足
执行int y = f(n-2)+n

stack 栈

int n = 3

int n = 5

int n = 5

第 8 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

n=3, f(n-2) = f(1)

stack 栈

int n = 3

int n = 5

int n = 5

第9步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

1. 栈内存中分配临时变量 int n = 1
2. 为方法 f(int n) 传入参数 n=1
3. 调用 f(1)
4. f(3) 挂起



第 10 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

n=1, if()条件满足, 返回调用者
此处为第8步中f(3)中调用了f(1)

stack 栈

int n = 1

int n = 3

int n = 5

int n = 5

注意：

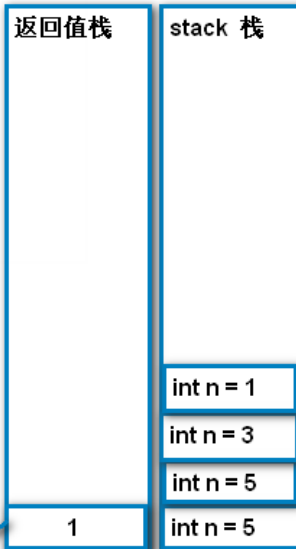
- ✓ 栈存储方式会有风险，以这种“对柴火”的方式存储数据会造成**栈溢出**，windows 平台下，JVM 默认的栈空间为 **64M 字节**，也可以通过优化参数修改，此处不做详解。

第 10 步补充说明

```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

在方法中一旦执行return语句，之后的代码就不再执行，方法结束

返回值存放在值栈中



注意：

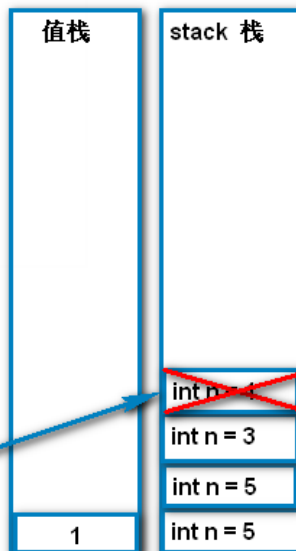
- ✓ 方法结束后返回值返回给调用者，JVM 开始回收临时变量。

第 11 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

1. 返回第8步中调用f(1)的地方
f(n-2) = f(1) = 1

2. f(1)返回，方法结束，JVM
回收f(1)中使用的临时变量
，int n = 1消失



第 12 步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

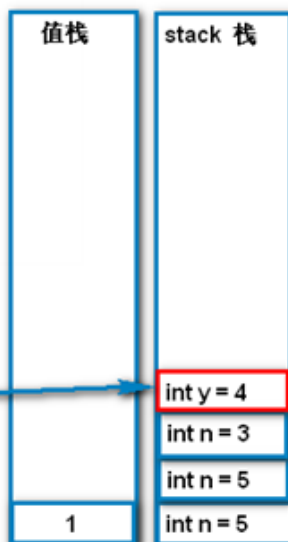
$f(n-2) + n = f(1) + 3 = 4$
注: n是从栈顶取出的
当前为3



第 13 步

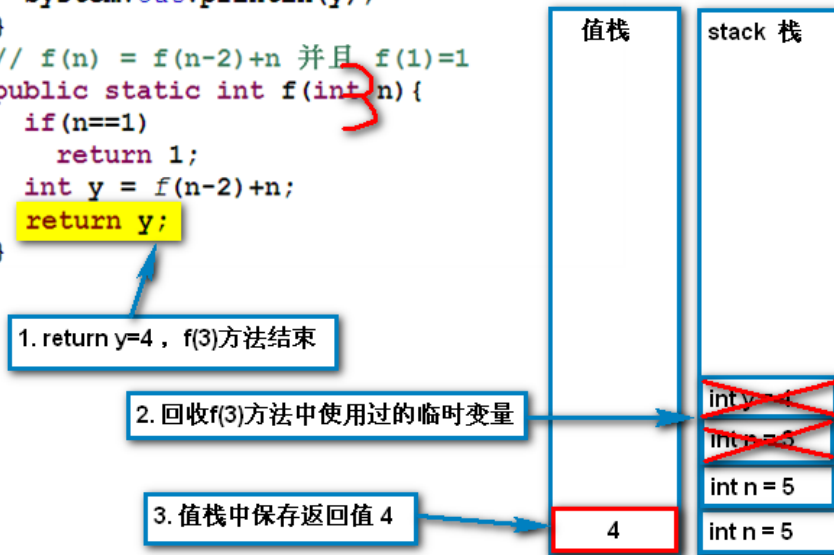
```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

栈内存中分配临时变量 int y, 将表达式 $f(n-2)+n$ 的值赋给 y



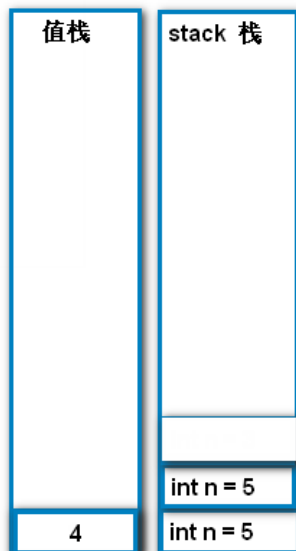
第 14 步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```



第 15 步

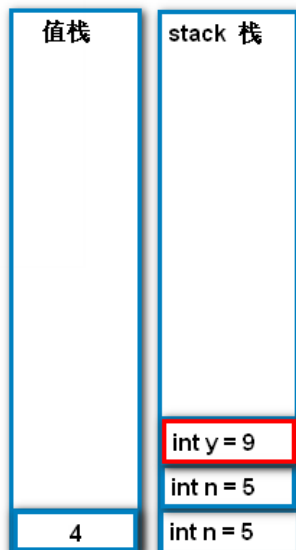
```
public static void main(String[] args) {
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```



返回到第5步中调用f(3)的地方
 $f(n-2) = f(3) = 4$

第 16 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n) ;//5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```



栈内存中分配临时变量int y,
将表达式 $f(n-2)+n$ 的值赋给y
 $y = f(n-2)+n = 4 + 5 = 9$

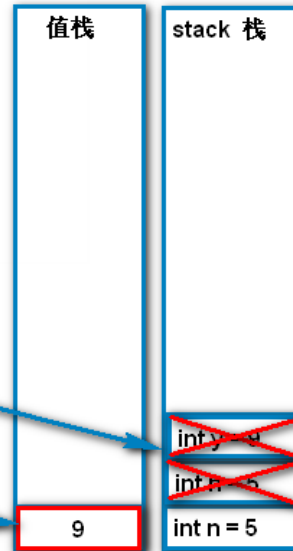
第 17 步

```
public static void main(String[] args){
    int n = 5;
    int y = f(n); //5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n){
    if(n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

1. return y=9, f(5)方法结束

2. 回收f(5)方法中使用过的临时变量

3. 值栈中保留返回值 9



第 18 步

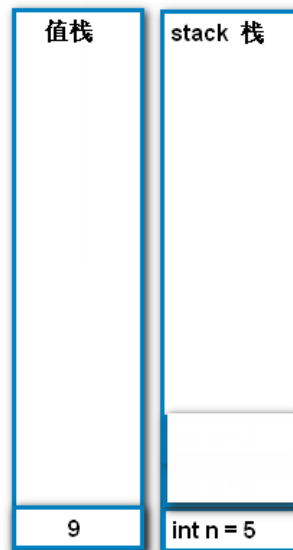
```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```

1. 返回给调用f(5)的位置 f(n) = 9
2. 栈内存中声明临时变量y, 并赋值



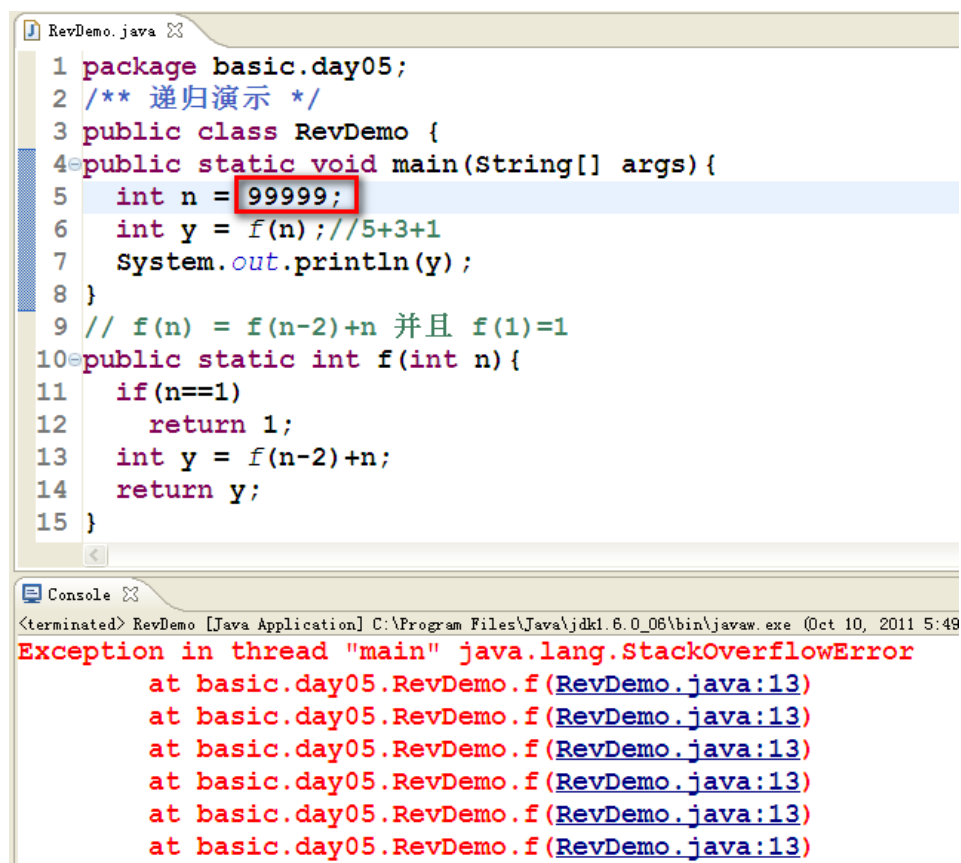
第 19 步

```
public static void main(String[] args) {
    int n = 5;
    int y = f(n); // 5+3+1
    System.out.println(y);
}
// f(n) = f(n-2)+n 并且 f(1)=1
public static int f(int n) {
    if (n==1)
        return 1;
    int y = f(n-2)+n;
    return y;
}
```



结束!

在使用递归调用时，如果递归次数过多会造成**栈溢出**现象，如图所示：



The screenshot shows a Java IDE with two windows. The top window, titled 'RevDemo.java', contains the following code:

```

1 package basic.day05;
2 /** 递归演示 */
3 public class RevDemo {
4     public static void main(String[] args){
5         int n = 99999;
6         int y = f(n); //5+3+1
7         System.out.println(y);
8     }
9     // f(n) = f(n-2)+n 并且 f(1)=1
10    public static int f(int n){
11        if(n==1)
12            return 1;
13        int y = f(n-2)+n;
14        return y;
15    }

```

The value '99999' on line 5 is highlighted with a red box. The bottom window, titled 'Console', shows the output of the program, which is a **StackOverflowError** exception:

```

<terminated> RevDemo [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Oct 10, 2011 5:49
Exception in thread "main" java.lang.StackOverflowError
    at basic.day05.RevDemo.f(RevDemo.java:13)
    at basic.day05.RevDemo.f(RevDemo.java:13)
    at basic.day05.RevDemo.f(RevDemo.java:13)
    at basic.day05.RevDemo.f(RevDemo.java:13)
    at basic.day05.RevDemo.f(RevDemo.java:13)
    at basic.day05.RevDemo.f(RevDemo.java:13)

```

递归调用过程中有**不断地在栈内存中声明变量**的特性，所以递归效率较低。