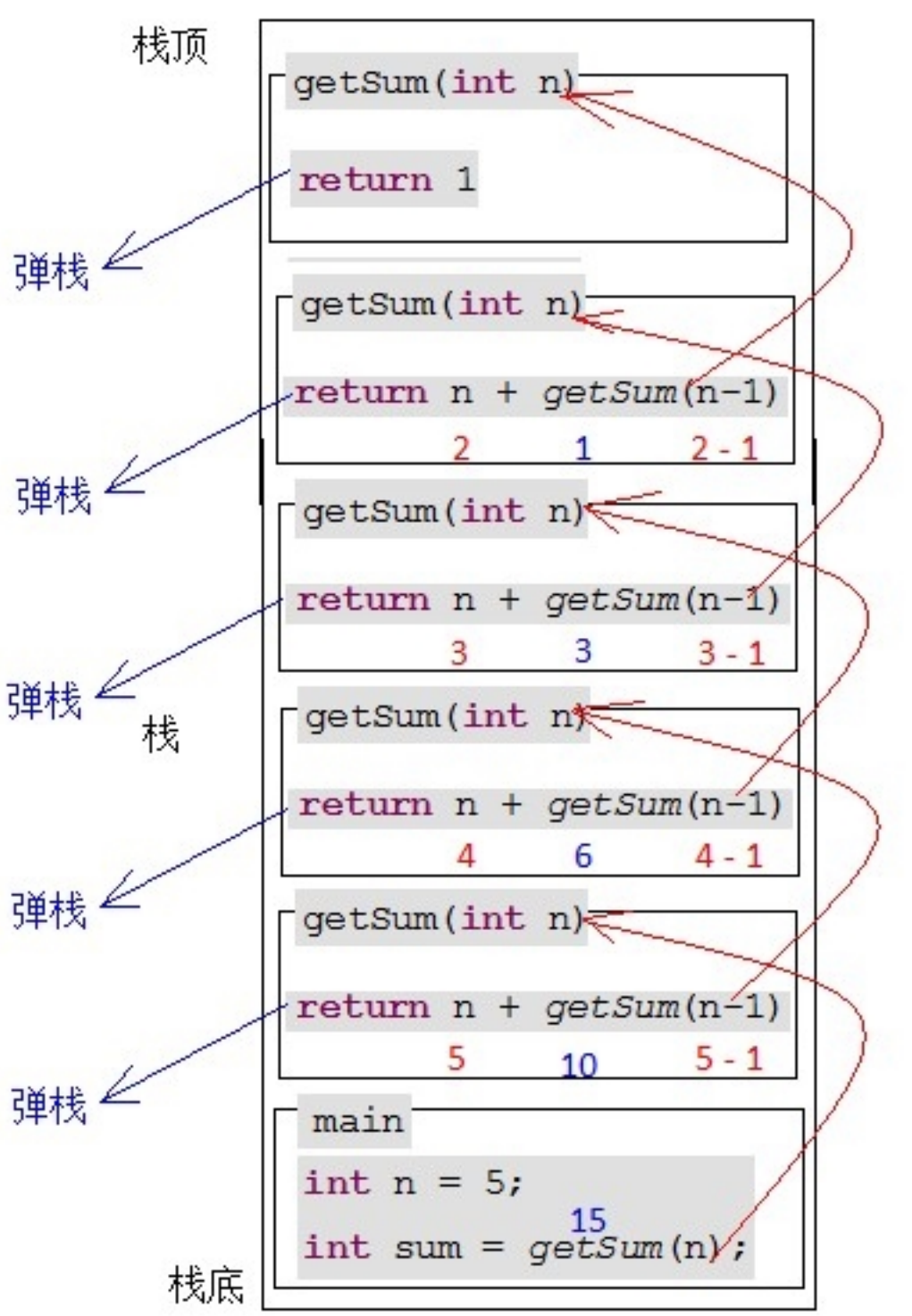
```java
public class DiGuiDemo {
    public static void main(String[] args) {
        //计算1~num的和，使用递归完成
        int n = 5;
        int sum = getSum(n);
        System.out.println(sum);

    }
    public static int getSum(int n) {
        if(n == 1){
            return 1;
        }
        return n + getSum(n-1);
    }
}
```



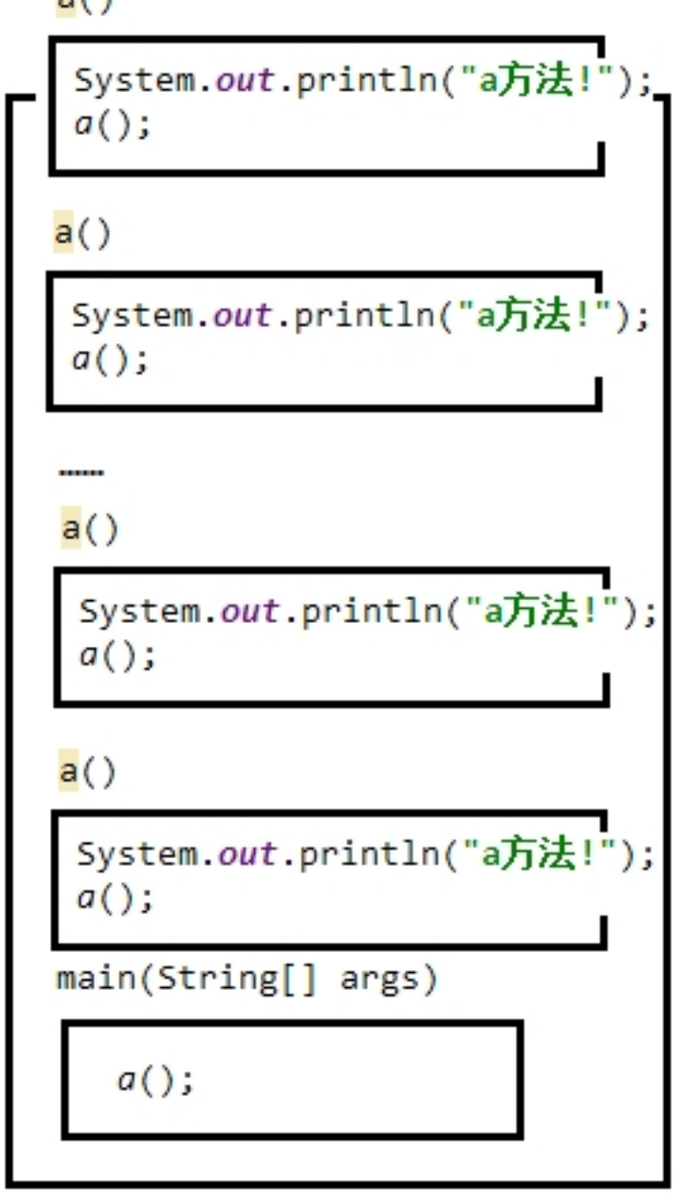
```java
public class Demo01Recurison {
    public static void main(String[] args) {
        a();
    }

    /*
        递归一定要有条件限定，保证递归能够停止下来，否则会发生栈内存溢出。
        Exception in thread "main" java.lang.StackOverflowError
     */
    private static void a() {
        System.out.println("a方法!");
        a();
    }
}
```

a方法会在栈内存中一直调用a方法,就会导致栈内存中有无数多个a方法

方法太多了,超出栈内存的大小,就会导致内存溢出的错误

注意:

当一个方法调用其他方法的时候,被调用的方法没有执行完毕,当前方法会一直等待调用的方法执行完毕,才会继续执行



栈内存

使用递归计算1-n之间和的原理

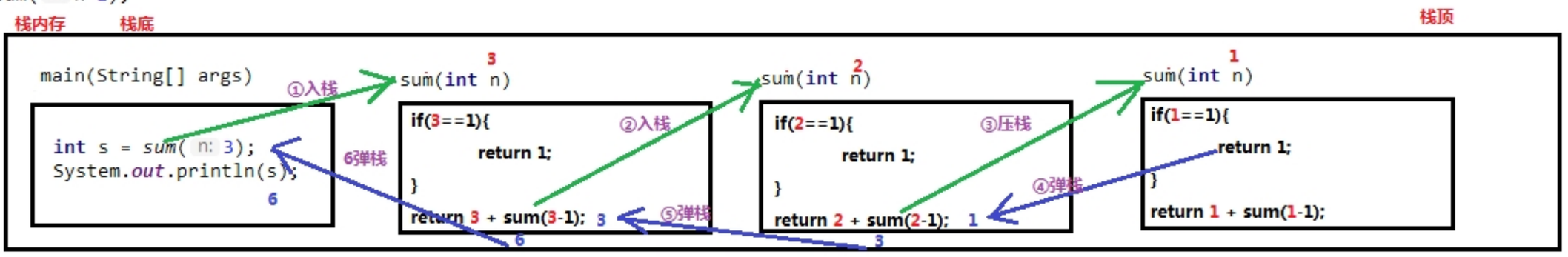
```
public class Demo02Recurison {
    public static void main(String[] args) {
        int s = sum( n: 3);
        System.out.println(s);
    }


public static int sum(int n){
    //获取到1的时候结束
    if(n==1){
        return 1;
    }

    //获取下一个被加的数字(n-1)
    return n + sum( n: n-1);
}
```

注意:

使用递归求和,main方法调用sum方法,sum方法会一直调用sum方法

导致在内存中有多个sum方法(频繁的创建方法,调用方法,销毁方法)效率低下

所以如果仅仅是计算1-n之间的和,不推荐使用递归,使用for循环即可



练习:

递归打印多级目录

需求:

遍历c:\\abc文件夹,及abc文件夹的子文件夹

```
c:\\abc
c:\\abc\\abc.txt
c:\\abc\\abc.java
c:\\abc\\a
c:\\abc\\a\\a.jpg
c:\\abc\\a\\a.java
c:\\abc\\b
c:\\abc\\b\\b.java
c:\\abc\\b\\b.txt
```

输出结果:

```
c:\abc\a
c:\abc\abc.java
c:\abc\abc.txt
c:\abc\b
```

```java
public class Demo04Recurison {
    public static void main(String[] args) {
        File file = new File( pathname: "c:\\abc");
        getAllFile(file);
    }

    /*
        定义一个方法,参数传递File类型的目录
        方法中对目录进行遍历
     */
    public static void getAllFile(File dir){
        File[] files = dir.listFiles();
        for (File f : files) {
            System.out.println(f);
        }
    }
}
```

分析:

发现遍历的结果并不完全,只有abc文件夹中的内容,没有子文件夹a和b的内容

有: a abc.java abc.txt b

没有: a.jpg a.java b.java b.txt

解决:

```
//对遍历得到的File对象f进行判断,判断是否是文件夹
if(f.isDirectory){
        //f是一个文件夹,则继续遍历这个文件夹
        //我们发现getAllFile方法就是传递文件夹,遍历文件夹的方法
        //所以直接调用getAllFile方法即可:递归(自己调用自己)
        getAllFile(f);
}else{
        //f是一个文件,直接打印即可
        System.out.println(f);
}
```

需求:

遍历c:\\abc文件夹,及abc文件夹的子文件夹

只要.java结尾的文件

c:\\abc
c:\\abc\\abc.txt
c:\\abc\\abc.java
c:\\abc\\a
c:\\abc\\a\\a.jpg
c:\\abc\\a\\a.java
c:\\abc\\b
c:\\abc\\b\\b.java
c:\\abc\\b\\b.txt

必须明确两件事情:

1.过滤器中的accept方法是谁调用的

2.accept方法的参数pathname是什么?

```java
public class Demo01Filter {
    public static void main(String[] args) {
        File file = new File( pathname: "c:\\abc");
        getAllFile(file);
    }

    public static void getAllFile(File dir){
        File[] files = dir.listFiles(new FileFilterImpl());//传递过滤器对象
        for (File f : files) {
            //对遍历得到的File对象f进行判断,判断是否是文件夹
            if(f.isDirectory()){
                //f是一个文件夹,则继续遍历这个文件夹
                //我们发现getAllFile方法就是传递文件夹,遍历文件夹的方法
                //所以直接调用getAllFile方法即可:递归(自己调用自己)
                getAllFile(f);
            }else{
                //f是一个文件,直接打印即可
                System.out.println(f);
            }
        }
    }
```

File
c:\\abc\\abc.txt

true

c:\\abc\\abc.txt true

accept方法返回值是一个布尔值

true:就会把传递过去的File对象保存到File数组中

false:就不会把传递过去的File对象保存到File数组中

```java
/*
    创建过滤器FileFilter的实现类,重写过滤方法accept,定义过滤规则
 */
public class FileFilterImpl implements FileFilter{
    @Override
    public boolean accept(File pathname) {
        return true;
    }
}
```

c:\\abc\\abc.txt

listFiles方法一共做了3件事情:

1.listFiles方法会对构造方法中传递的目录进行遍历,获取目录中的每一个文件/文件夹-->封装为File对象

2.listFiles方法会调用参数传递的过滤器中的方法accept

3.listFiles方法会把遍历得到的每一个File对象,传递过accept方法的参数pathname

过滤的规则:

在accept方法中,判断File对象是否是以.java结尾

是就返回true

不是就返回false

day08【File类、递归】

主要内容

- File类
- 递归

教学目标

- [ ] 能够说出File对象的创建方式
- [ ] 能够说出File类获取名称的方法名称
- [ ] 能够说出File类获取绝对路径的方法名称
- [ ] 能够说出File类获取文件大小的方法名称
- [ ] 能够说出File类判断是否是文件的方法名称
- [ ] 能够说出File类判断是否是文件夹的方法名称
- [ ] 能够辨别相对路径和绝对路径
- [ ] 能够遍历文件夹
- [ ] 能够解释递归的含义
- [ ] 能够使用递归的方式计算5的阶乘
- [ ] 能够说出使用递归会内存溢出隐患的原因

第一章 File类

1.1 概述

`java.io.File` 类是文件和目录路径名的抽象表示，主要用于文件和目录的创建、查找和删除等操作。

1.2 构造方法

- `public File(String pathname)` ：通过将给定的**路径名字符串**转换为抽象路径名来创建新的 File实例。
- `public File(String parent, String child)` ：从**父路径名字符串和子路径名字符串**创建新的 File实例。
- `public File(File parent, String child)` ：从**父抽象路径名和子路径名字符串**创建新的 File实例。

- 构造举例，代码如下：

```
// 文件路径名
String pathname = "D:\\aaa.txt";
File file1 = new File(pathname);

// 文件路径名
String pathname2 = "D:\\aaa\\bbb.txt";
File file2 = new File(pathname2);
```

```
// 通过父路径和子路径字符串
 String parent = "d:\\aaa";
 String child = "bbb.txt";
 File file3 = new File(parent, child);

// 通过父级File对象和子路径字符串
File parentDir = new File("d:\\aaa");
String child = "bbb.txt";
File file4 = new File(parentDir, child);
```

> 小贴士：
>
> 1. 一个File对象代表硬盘中实际存在的一个文件或者目录。
> 2. 无论该路径下是否存在文件或者目录，都不影响File对象的创建。

1.3 常用方法

获取功能的方法

- `public String getAbsolutePath()` ：返回此File的绝对路径名字符串。
- `public String getPath()` ：将此File转换为路径名字符串。
- `public String getName()` ：返回由此File表示的文件或目录的名称。
- `public long length()` ：返回由此File表示的文件的长度。

方法演示，代码如下：

```
public class FileGet {
    public static void main(String[] args) {
        File f = new File("d:/aaa/bbb.java");
        System.out.println("文件绝对路径:"+f.getAbsolutePath());
        System.out.println("文件构造路径:"+f.getPath());
        System.out.println("文件名称:"+f.getName());
        System.out.println("文件长度:"+f.length()+"字节");

        File f2 = new File("d:/aaa");
        System.out.println("目录绝对路径:"+f2.getAbsolutePath());
        System.out.println("目录构造路径:"+f2.getPath());
        System.out.println("目录名称:"+f2.getName());
        System.out.println("目录长度:"+f2.length());
    }
}
输出结果：
文件绝对路径:d:\aaa\bbb.java
文件构造路径:d:\aaa\bbb.java
文件名称:bbb.java
文件长度:636字节

目录绝对路径:d:\aaa
目录构造路径:d:\aaa

目录名称:aaa
```

```
目录长度:4096
```

> API中说明：length()，表示文件的长度。但是File对象表示目录，则返回值未指定。

绝对路径和相对路径

- **绝对路径**：从盘符开始的路径，这是一个完整的路径。
- **相对路径**：相对于项目目录的路径，这是一个便捷的路径，开发中经常使用。

```
public class FilePath {
    public static void main(String[] args) {
        // D盘下的bbb.java文件
        File f = new File("D:\\bbb.java");
        System.out.println(f.getAbsolutePath());

         // 项目下的bbb.java文件
        File f2 = new File("bbb.java");
        System.out.println(f2.getAbsolutePath());
    }
}
输出结果：
D:\bbb.java
D:\idea_project_test4\bbb.java
```

判断功能的方法

- `public boolean exists()` ：此File表示的文件或目录是否实际存在。
- `public boolean isDirectory()` ：此File表示的是否为目录。
- `public boolean isFile()` ：此File表示的是否为文件。

方法演示，代码如下：

```
public class FileIs {
    public static void main(String[] args) {
        File f = new File("d:\\aaa\\bbb.java");
        File f2 = new File("d:\\aaa");
        // 判断是否存在
        System.out.println("d:\\aaa\\bbb.java 是否存在:"+f.exists());
        System.out.println("d:\\aaa 是否存在:"+f2.exists());
        // 判断是文件还是目录
        System.out.println("d:\\aaa 文件?:"+f2.isFile());
        System.out.println("d:\\aaa 目录?:"+f2.isDirectory());
    }
}
输出结果：
d:\aaa\bbb.java 是否存在:true
d:\aaa 是否存在:true
d:\aaa 文件?:false
d:\aaa 目录?:true
```

创建删除功能的方法

- `public boolean createNewFile()` ：当且仅当具有该名称的文件尚不存在时，创建一个新的空文件。
- `public boolean delete()` ：删除由此File表示的文件或目录。
- `public boolean mkdir()` ：创建由此File表示的目录。
- `public boolean mkdirs()` ：创建由此File表示的目录，包括任何必需但不存在的父目录。

方法演示，代码如下：

```
public class FileCreateDelete {
    public static void main(String[] args) throws IOException {
        // 文件的创建
        File f = new File("aaa.txt");
        System.out.println("是否存在:"+f.exists()); // false
        System.out.println("是否创建:"+f.createNewFile()); // true
        System.out.println("是否存在:"+f.exists()); // true

        // 目录的创建
        File f2= new File("newDir");
        System.out.println("是否存在:"+f2.exists());// false
        System.out.println("是否创建:"+f2.mkdir()); // true
        System.out.println("是否存在:"+f2.exists());// true

         // 创建多级目录
        File f3= new File("newDira\\newDirb");
        System.out.println(f3.mkdir());// false
        File f4= new File("newDira\\newDirb");
        System.out.println(f4.mkdirs());// true

        // 文件的删除
        System.out.println(f.delete());// true

        // 目录的删除
        System.out.println(f2.delete());// true
        System.out.println(f4.delete());// false
    }
}
```

> API中说明：delete方法，如果此File表示目录，则目录必须为空才能删除。

1.4 目录的遍历

- `public String[] list()` ：返回一个String数组，表示该File目录中的所有子文件或目录。
- `public File[] listFiles()` ：返回一个File数组，表示该File目录中的所有的子文件或目录。

```
public class FileFor {
    public static void main(String[] args) {
        File dir = new File("d:\\java_code");

        //获取当前目录下的文件以及文件夹的名称。
         String[] names = dir.list();
         for(String name : names){

             System.out.println(name);
```

```
        }
        //获取当前目录下的文件以及文件夹对象，只要拿到了文件对象，那么就可以获取更多信息
        File[] files = dir.listFiles();
        for (File file : files) {
            System.out.println(file);
        }
    }
}
```

> 小贴士：
>
> 调用listFiles方法的File对象，表示的必须是实际存在的目录，否则返回null，无法进行遍历。

第二章 递归

2.1 概述

- **递归**：指在当前方法内调用自己的这种现象。
- **递归的分类:**
  - 递归分为两种，直接递归和间接递归。
  - 直接递归称为方法自身调用自己。
  - 间接递归可以A方法调用B方法，B方法调用C方法，C方法调用A方法。
- **注意事项**：
  - 递归一定要有条件限定，保证递归能够停止下来，否则会发生栈内存溢出。
  - 在递归中虽然有限定条件，但是递归次数不能太多。否则也会发生栈内存溢出。
  - 构造方法,禁止递归

```
public class Demo01DiGui {
    public static void main(String[] args) {
        // a();
        b(1);
    }

    /*
     * 3.构造方法,禁止递归
     * 编译报错:构造方法是创建对象使用的,不能让对象一直创建下去
     */
    public Demo01DiGui() {
        //Demo01DiGui();
    }


    /*
     * 2.在递归中虽然有限定条件，但是递归次数不能太多。否则也会发生栈内存溢出。
     * 4993
     *  Exception in thread "main" java.lang.StackOverflowError
     */
    private static void b(int i) {

        System.out.println(i);
```

```
        //添加一个递归结束的条件,i==5000的时候结束
        if(i==5000){
            return;//结束方法
        }
        b(++i);
    }

    /*
     * 1.递归一定要有条件限定，保证递归能够停止下来，否则会发生栈内存溢出。 Exception in thread "main"
     * java.lang.StackOverflowError
     */
    private static void a() {
        System.out.println("a方法");
        a();
    }
}
```

2.2 递归累加求和

计算1 ~ n的和

分析：num的累和 = num + (num-1)的累和，所以可以把累和的操作定义成一个方法，递归调用。

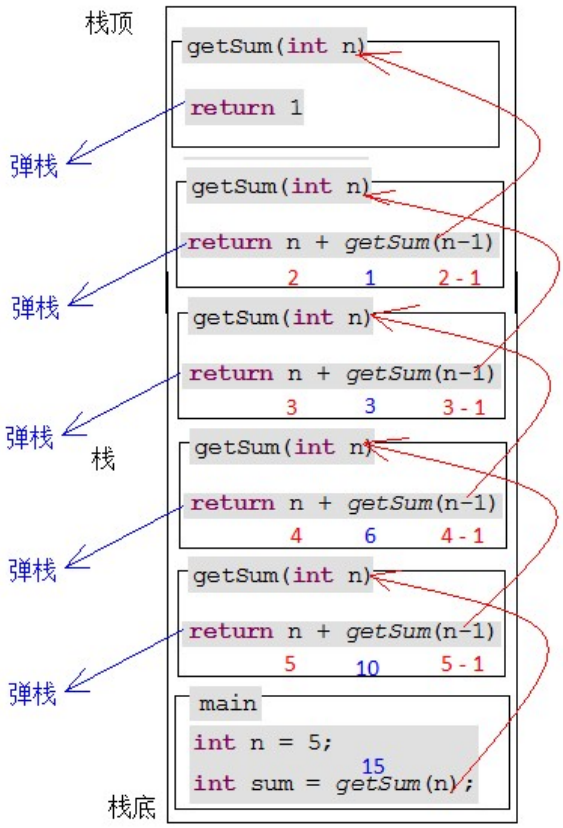
实现代码：

```
public class DiGuiDemo {
    public static void main(String[] args) {
        //计算1~num的和，使用递归完成
        int num = 5;
        // 调用求和的方法
        int sum = getSum(num);
        // 输出结果
        System.out.println(sum);

    }
    /*
      通过递归算法实现.
      参数列表:int
      返回值类型: int
    */
    public static int getSum(int num) {
        /*
           num为1时,方法返回1,
           相当于是方法的出口,num总有是1的情况
        */
        if(num == 1){
            return 1;
        }
        /*
          num不为1时,方法返回 num +(num-1)的累和
          递归调用getSum方法

        */
```

```
        return num + getSum(num-1);
    }
}
```

代码执行图解

```
public class DiGuiDemo {
    public static void main(String[] args) {
        //计算1~num的和，使用递归完成
        int n = 5;
        int sum = getSum(n);
        System.out.println(sum);

    }
    public static int getSum(int n) {
        if(n == 1){
            return 1;
        }
        return n + getSum(n-1);
    }
}
```



> 小贴士：递归一定要有条件限定，保证递归能够停止下来，次数不要太多，否则会发生栈内存溢出。

2.3 递归求阶乘

- **阶乘**：所有小于及等于该数的正整数的积。

```
n的阶乘：n! = n * (n-1) *...* 3 * 2 * 1
```

分析：这与累和类似,只不过换成了乘法运算，学员可以自己练习，需要注意阶乘值符合int类型的范围。

```
推理得出：n! = n * (n-1)!
```

代码实现：

```
public class DiGuiDemo {
    //计算n的阶乘，使用递归完成
    public static void main(String[] args) {
        int n = 3;
        // 调用求阶乘的方法
        int value = getValue(n);
        // 输出结果
```

```
        System.out.println("阶乘为:"+ value);
    }
    /*
      通过递归算法实现.
      参数列表:int
      返回值类型: int
    */
    public static int getValue(int n) {
        // 1的阶乘为1
        if (n == 1) {
            return 1;
        }
        /*
          n不为1时,方法返回 n! = n*(n-1)!
          递归调用getValue方法
        */
        return n * getValue(n - 1);
    }
}
```

2.4 递归打印多级目录

分析：多级目录的打印，就是当目录的嵌套。遍历之前，无从知道到底有多少级目录，所以我们还是要使用递归实现。

代码实现：

```
public class DiGuiDemo2 {
    public static void main(String[] args) {
        // 创建File对象
        File dir  = new File("D:\\aaa");
        // 调用打印目录方法
        printDir(dir);
    }

    public static void  printDir(File dir) {
        // 获取子文件和目录
        File[] files = dir.listFiles();
        // 循环打印
        /*
          判断:
          当是文件时,打印绝对路径.
          当是目录时,继续调用打印目录的方法,形成递归调用.
        */
        for (File file : files) {
            // 判断
            if (file.isFile()) {
                // 是文件,输出文件绝对路径
                System.out.println("文件名:"+ file.getAbsolutePath());
            } else {
                // 是目录,输出目录绝对路径

                System.out.println("目录:"+file.getAbsolutePath());
```

```
                // 继续遍历,调用printDir,形成递归
                printDir(file);
            }
        }
    }
}
```

第三章 综合案例

3.1 文件搜索

搜索`D:\aaa`目录中的`.java`文件。

分析：

1. 目录搜索，无法判断多少级目录，所以使用递归，遍历所有目录。
2. 遍历目录时，获取的子文件，通过文件名称，判断是否符合条件。

代码实现：

```
public class DiGuiDemo3 {
    public static void main(String[] args) {
        // 创建File对象
        File dir  = new File("D:\\aaa");
        // 调用打印目录方法
        printDir(dir);
    }

    public static void printDir(File dir) {
        // 获取子文件和目录
        File[] files = dir.listFiles();

        // 循环打印
        for (File file : files) {
            if (file.isFile()) {
                // 是文件，判断文件名并输出文件绝对路径
                if (file.getName().endsWith(".java")) {
                    System.out.println("文件名:" + file.getAbsolutePath());
                }
            } else {
                // 是目录，继续遍历,形成递归
                printDir(file);
            }
        }
    }
}
```

3.2 文件过滤器优化

`java.io.FileFilter`是一个接口，是File的过滤器。 该接口的对象可以传递给File类的`listFiles(FileFilter)`作为参数， 接口中只有一个方法。

`boolean accept(File pathname)` ：测试pathname是否应该包含在当前File目录中，符合则返回true。

分析：

1. 接口作为参数，需要传递子类对象，重写其中方法。我们选择匿名内部类方式，比较简单。
2. `accept`方法，参数为File，表示当前File下所有的子文件和子目录。保留住则返回true，过滤掉则返回false。保留规则：
    1. 要么是.java文件。
    2. 要么是目录，用于继续遍历。
3. 通过过滤器的作用，`listFiles(FileFilter)`返回的数组元素中，子文件对象都是符合条件的，可以直接打印。

代码实现：

```
public class DiGuiDemo4 {
    public static void main(String[] args) {
        File dir = new File("D:\\aaa");
        printDir2(dir);
    }

    public static void printDir2(File dir) {
        // 匿名内部类方式,创建过滤器子类对象
        File[] files = dir.listFiles(new FileFilter() {
            @Override
            public boolean accept(File pathname) {
                return pathname.getName().endsWith(".java")||pathname.isDirectory();
            }
        });
        // 循环打印
        for (File file : files) {
            if (file.isFile()) {
                System.out.println("文件名:" + file.getAbsolutePath());
            } else {
                printDir2(file);
            }
        }
    }
}
```

3.3 Lambda优化

分析：`FileFilter`是只有一个方法的接口，因此可以用lambda表达式简写。

lambda格式：

```
()->{ }
```

代码实现：

```java
public static void printDir3(File dir) {
    // lambda的改写
    File[] files = dir.listFiles(f ->{
        return f.getName().endsWith(".java") || f.isDirectory();
    });

    // 循环打印
    for (File file : files) {
        if (file.isFile()) {
            System.out.println("文件名:" + file.getAbsolutePath());
        } else {
            printDir3(file);
        }
    }
}
```