**案例1 for循环嵌套**

**一、案例描述**

1. 考核知识点

编号：00102018

名称：循环嵌套

1. 练习目标

* 掌握循环嵌套的使用和执行顺序

1. 需求分析

在编程过程中，经常需要双重循环或多重循环，这就需要循环语句的嵌套。最常见的循环嵌套就是在for循环中嵌套for循环。为了让初学者熟悉for循环嵌套的使用，本案例将演示如何用for循环嵌套打印由“\*”组成的直角三角形。具体要求如下：

* 第一个三角形为倒直角三角形，共5行，第一行5个 “\*”，一行减少一个，第5行一个“\*”
* 将倒直角三角形上下翻转为正直角三角形，第一行1一个“\*”，第5行5个“\*”

1. 设计思路（实现原理）
2. 分析第一个三角形的打印，外层for循环控制行数，初始化表达式从0开始，循环条件为小于5；第一行是5个“\*”所以应该内层循环打印5次，初始化表达式从0开始，循环条件为小于5。同理，内层的循环初始化和条件如下所示：

0 1 2 3 4

1 2 3 4

2 3 4

3 4

4

由此可见内层循环的初始化表达式y 与外层的x是相等的，循环条件都是小于5。

1. 分析第二个三角形的打印，第一行1个“\*”，第二行2个“\*”...由此可见，内层循环的循环次数与行数是相等的，所以内层循环的y小于或等于外层的x。

**二、案例实现**

定义Example15类，代码如下所示:

class Example15

{

public static void main(String[] args)

{

for(int x = 0 ; x < 5 ; x++)

{

for(int y = x; y < 5 ; y++)

{

System.out.print("\*");

}

System.out.println();

}

System.out.println("--------------------------");

for(int x = 0 ; x < 5 ; x++)

{

for(int y = 0; y <= x ; y++)

{

System.out.print("\*");

}

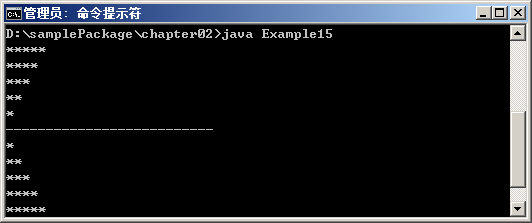
System.out.println();

}

}

}

运行结果如图2-18所示。



1. 运行结果

**三、案例总结**

由于嵌套循环程序比较复杂，下面分步骤进行讲解，具体如下：

第一个三角形：

1. 当x=0时，y=0且y<5，循环5次，完成后返回到外层循环进行换行。
2. 当x=1时，y=1且y<5，循环4次，完成后返回到外层循环换行。
3. 当x=2时，y=2，循环3次，完成后返回到外层循环换行。
4. 当x=3时，y=3，循环2次，完成后返回到外层循环换行。
5. 当x=4时，y=4，循环1次，完成后返回到外层循环换行。由于x<5所以循环终止，打印结束。

第二个三角形：

1. 当x=0时，y=0且y<=0，循环1次，完成后返回到外层循环进行换行。
2. 当x=1时，y=0且y<=1，循环2次，完成后返回到外层循环进行换行。
3. 当x=2时，y=0且y<=2，循环3次，完成后返回到外层循环进行换行。
4. 当x=3时，y=0且y<=3，循环4次，完成后返回到外层循环进行换行。
5. 当x=4时，y=0且y<=4，循环5次，完成后返回到外层循环进行换行。由于x<5所以循环终止，打印结束。

## 案例2 面向对象

1. 定义一个抽象的"Role"类 ，成员变量有name， age， gender等变量，要求尽可能隐藏所有变量(能够私有就私有,能够保护就不要公有) 再通过GetXXX()和SetXXX()方法对各变量进行读写。具有一个抽象的play()方法， 该方法不返回任何值 ，同时定义一个无参构造方法和一个有参构造方法，有参构造方法中要体现出this用法。

2. 从Role类派生出一个"Employee"类 该类具有Role类的所有成员， 构造方法除外， 并增加一个 salary成员变量 和一个id成员变量。同样提供一个无参构造方法和一个有参构造方法，有参构造方法中要体现出this和super的用法 。要求重写play()方法 ，并提供一个final sing()方法。

3. 创建一个测试类Test，在main()方法中创建Employee对象,并测试该对象的方法。

示例代码：

**abstract** **class** Role

{

**protected** String name;

**protected** **int** age;

**protected** String sex;

**public** Role(){}

**public** Role(String name,**int** age,String sex)

{

**this**.name=name;

**this**.age=age;

**this**.sex=sex;

}

**public** **abstract** **void** play();

}

**class** Employee **extends** Role

{

**protected** **int** id;

**protected** **int** salary;

**public** Employee(){}

**public** Employee(String name,**int** age,String sex,**int** id,**int** salary)

{

**super**(name,age,sex);

**this**.id=id;

**this**.salary=salary;

}

**public** **void** play()

{

System.***out***.println("i can paly");

}

**public** **final** **void** sing()

{

System.***out***.println("i can sing");

}

}

**public** **class** Test {

**public** **static** **void** main(String[] dsa)

{

Employee e=**new** Employee();

e.id=1201012204;

e.salary=15000;

e.name="you xiao";

e.age=20;

e.sex="Man";

e.play();

e.sing();

System.***out***.println(e.id);

System.***out***.println(e.salary);

System.***out***.println(e.name);

System.***out***.println(e.age);

System.***out***.println(e.sex);

}

}

## 案例3 自定义缓冲区拷贝大文件

### 一、案例描述

1. 考核知识点

编 号：00108005

名 称: 字节流的缓冲区

1. 练习目标

* 掌握如何自定义缓冲区
* 掌握如何使用自定义缓冲区读、写文件
* 了解自定义缓冲区读写的优点

1. 需求分析

操作文件时一个个字节的读写，效率非常低，为了提高效率可以自定义一个字节数组作为缓冲区。为了让初学者掌握如何使用自定义缓冲区读写文件，案例中将演示使用字节流的自定义缓冲区拷贝“IO流.avi”文件到其它目录来学习。

1. 设计思路（实现原理）
2. 在当前目录下创建source和target文件夹，并在source文件夹中存放一个“IO流.avi”文件。
3. 编写Example03类。
4. 在main()方法中，分别创建文件字符输出流和文件字符输入流对象，并指定文件拷贝的源及目标路径。
5. 自定义一个1024长度的字节数组作为缓冲区，然后通过输入流将源文件数据读取到缓冲区中，输出流再将缓冲区中数据写入到目标文件中，循环往复直到文件拷贝结束，并计算拷贝文件总耗时。
6. 使用close()方法，关闭流资源。

### 二、案例实现

import java.io.\*;

public class Example03 {

public static void main(String[] args) {

FileInputStream fis = null;

FileOutputStream fos = null;

try {

// 创建文件字符输出流和文件字符输入流对象

fis = new FileInputStream("source\\IO流.avi");

fos = new FileOutputStream("target\\IO流.avi");

// 创建字节数组，用于缓存读取的字节

byte[] bys = new byte[1024];

int len = 0;

long starttime = System.currentTimeMillis();// 获取拷贝文件前的系统时间

// 读取到文件结束时，返回-1

while ((len = fis.read(bys)) != -1) {

// 将缓冲区中的数据写入文件

fos.write(bys, 0, len);

}

long overtime = System.currentTimeMillis();// 获取拷贝文件结束的系统时间

System.out.println("拷贝文件消耗的时间为：" + (overtime - starttime) + "毫秒");

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("拷贝文件失败！");

} finally {

try {

// 关闭流资源

fis.close();

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("输入流关闭失败！");

}

try {

// 关闭流资源

fos.close();

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("输出流关闭失败！");

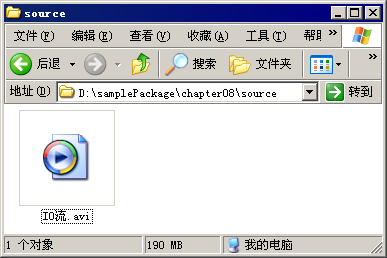
}

}

}

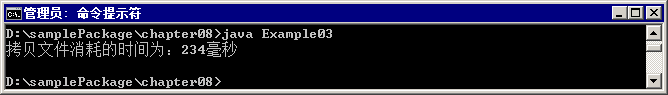
}

程序运行结束后，打开target文件夹，发现source文件夹中的“IO流.avi”文件被成功拷贝到了target文件夹，如图8-4所示。

1. 拷贝结果

程序运行结束后，会在命令行窗口输出拷贝文件所消耗的时间，如图8-5所示：



1. 运行结果

从运行结果可以看出，使用自定义缓冲区拷贝大文件成功，而且耗时较短。

### 三、案例总结

1、由于计算机性能等各方面原因，会导致每次拷贝文件所消耗的时间未必相同。

2、在指定文件路径时使用了“\\”，这是因为在Windows目录路径基础上使用了转义符“\”。

3、为了保证IO流的close()方法必须执行，通常将关闭流的操作写在finally代码块中。

4、思考一下：使用自定义缓冲区拷贝文件时效率高耗时短，请修改Example03类，在不使用缓冲区的情况下拷贝文件，打印所消耗的时间，并与使用缓冲区所耗时间比较。代码如下：

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class Example03 {

public static void main(String[] args) {

FileInputStream fis = null;

FileOutputStream fos = null;

try {

// 创建文件字符输出流和文件字符输入流对象

fis = new FileInputStream("source\\IO流.avi");

fos = new FileOutputStream("target\\IO流.avi");

// 创建字节数组，用于缓存读取的字节

int len = 0;

long starttime = System.currentTimeMillis();// 获取拷贝文件前的系统时间

// 读取到文件结束时，返回-1

while ((len = fis.read()) != -1) {

// 将缓冲区中的数据写入文件

fos.write(len);

}

long overtime = System.currentTimeMillis();// 获取拷贝文件结束的系统时间

System.out.println("拷贝文件消耗的时间为：" + (overtime - starttime) + "毫秒");

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("拷贝文件失败！");

} finally {

try {

// 关闭流资源

fis.close();

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("输入流关闭失败！");

}

try {

// 关闭流资源

fos.close();

} catch (IOException e) {

e.printStackTrace();

throw new RuntimeException("输出流关闭失败！");

}

}

}

}