2015. java学习交流群

8918 1289

每天有免费的Java学习课堂

——学习Java就是这么简单

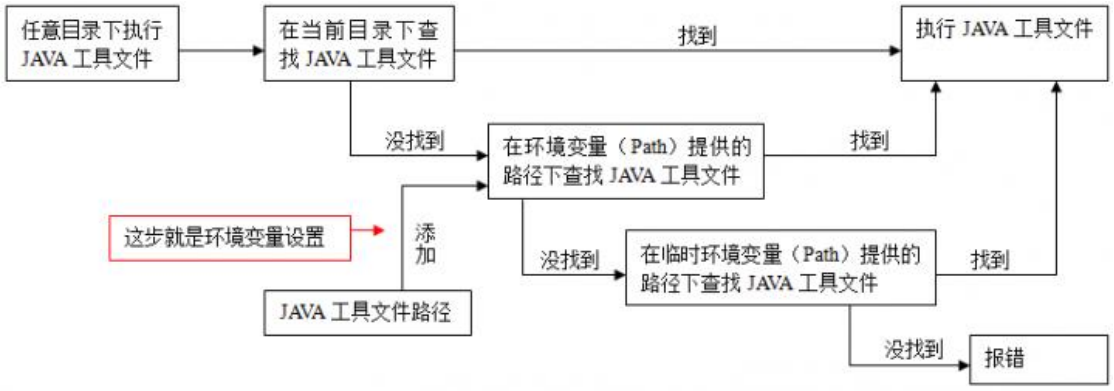
——为Java而燃烧——

# 目录

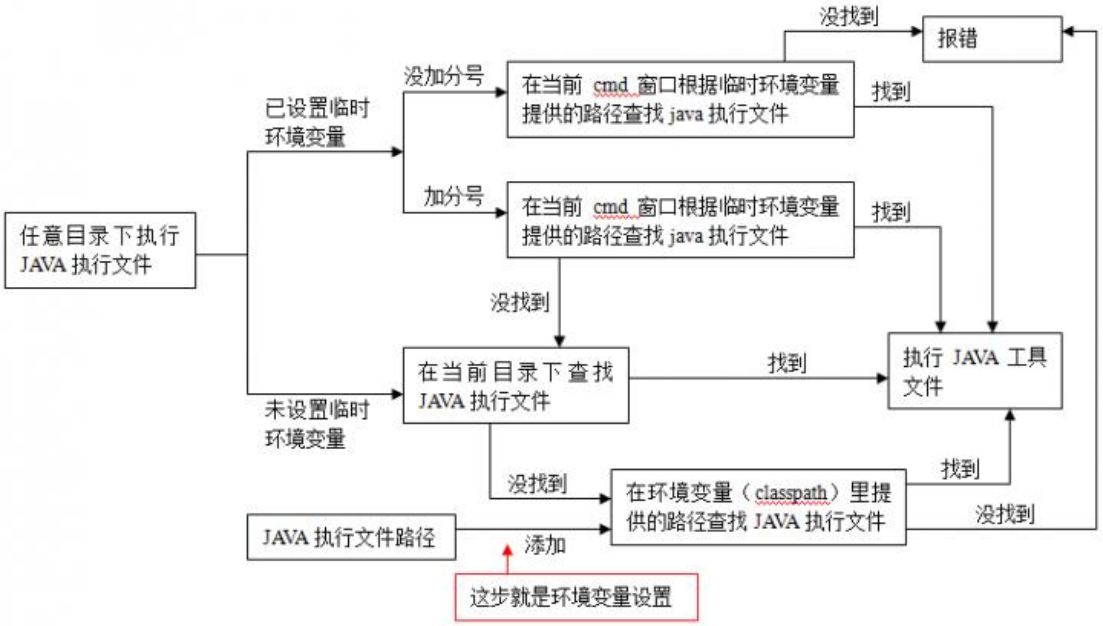
# 一、运行程序之前：

## 1.Path 与 classpath 图解

答案：path 顺序：




classpath 顺序：




## 2. Unicode 问题

答案 ：Unicode 规范中推荐的标记字节顺序的方法是 BOM。

①所以在转换为 Unicode 是加入了 BOM (Byte Order Mark) 可以理解为采用那种排序方式吧。

②如果接收者收到 FEFF(-1,-2)，就表明这个字节流是 Big-Endian 的；如果收到 FFFE(-2,-1)，就表明这个字节流是 Little-Endian 的。

③在 Java 中直接使用 Unicode 转码时会按照 UTF-16LE 的方式拆分 ,并加上 BOM 所以就是 FFFE(-2,-1)，这就是为什么会有-2,-1 了。

④UTF8 字节没有顺序，所以它可以被用来检测一个字节流是否是 UTF-8 编码的，所以在 UTF-8 中没有这个内容，我们平时用 UTF-8 比较多，所以看不到这个。

## 3.Eclipse 简便设置

答案：①Eclipse 中代码编辑背景颜色修改：代码编辑界面默认颜色为白色。对于长期使用

电脑编程的人来说，白色很刺激我们的眼睛，所以改变 workspace 的背景色，可以使眼睛舒服一些。设置方法如下：

A）打开 window / Preference,弹出 Preference 面板

B）展开 General 标签，选中 Editors 选项，展开。

C）选中 Text Editors，右边出现 TestEditors 面板。 面板中有这样一个选项：Appearance color options；其中是各种板块颜色的设置，其中有一项是 background color，勾掉 System Default，点击'color'，弹出颜色选择面板，选择喜好的颜色，单击确定。

注：背景颜色眼科专家推荐：色调 85，饱和度，123，亮度 205。文档都不再是刺眼的白底黑字，而是非常柔和的豆沙绿色，这个色调是眼科专家配置的，长时间使用会很有效的缓解眼睛疲劳保护眼睛。

D）返回 Test Editors，单击 Apply 即可。展开 TestEditors，还有其他选项，比如对错误提示的颜色样式，可根据需要尝试更改一下。

②Eclipse 字体大小调整：Window / Preferences / General / Appearance /ColorsAnd Fonts ，在右边的对话框里选择 Java- Java Editor Text Font，点击出现的编辑（Edit）按钮，可以设置显示在在主窗体中程序的字体大小，设置完之后点击右下角的应用(Apply)，最后点击确定（OK）即可。

Xml 文件字体大小的调整： window / preferences / General / appearance / colors and fonts /Basic / "Text font " ，然后点击 Edit，可以设置字体的大小。

注：上述的方法只是在 Eclipse 的一个简单的背景方案设置，Eclipse 里还可以使用专门的插件对代码背景进行设置，可查看《Eclipse 插件：Eclipse Color Theme》一文。

eclipse 更改 xml 文件,txt 文件,property 文件等文件编辑器的字体设置:window--preferences--General--appearance--colors and fonts--Basic-- "Text font "

③ Eclipse 代码自动补全

在做 Eclipse 项目的时候，发现代码补全功能不太好，总是需要在点击 “.” 号之后才能出现代码提示的功能。不想 Visual Studio 里面点击一个字母就出现代码提示。当然 Visual Studio 里面如果增加了 Visual Assistant 那就更加如虎添翼了。所以 google 了一下 Eclipse 里面怎么也实现和 Visual Studiao 一样的代码补全功能。下面是搜索到的文章。

打开 Eclipse-> Window -> Perferences，会打开个 Perferences 的设置界面.

找到 Java 下的 Editor 下的 Content Assist， 右边出现的选项中，有一个 Auto activationtriggers for Java:会看到只有一个"."存在。表示：只有输入"."之后才会有代码提示，我们要修改的地方就是这里，将 “.” 改为 “.abcdefghijklmnopqrstuvwxyz” 即可。

## 二、基础知识：

1.主方法中 args 是什么意思

答案：String[] args ：是保存运行 main 函数时输入的参数的字符串数组，当在 cmd 运行时，输入： java test a b c，数组就会将 abc 保存起来： args[0] = a; args[1] = b; args[2] = c;这些变量在程序中是可以调用的

2.&和&&的区别

答案：①&和&&都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。

②&&还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式，例如，对于 if(str != null && !str.equals(“”))表达式，当 str 为 null 时，后面的表达式不会执行，所以不会出现 NullPointerException 如果将&&改为&，则会抛出 NullPointerException 异常。If(x==33 & ++y>0) y 会增长，If(x==33 && ++y>0)不会增长。

③&还可以用作位运算符，当&操作符两边的表达式不是 boolean 类型时，&表示按位与操作，我们通常使用 0x0f 来与一个整数进行&运算，来获取该整数的最低 4 个 bit 位，例如，0x01 & 0x0f 的结果为 0x01。

3.Byte 类型的-128 怎么表示

答案：byte 用八位二进制码表示一个十进制数 ，所以范围是-128~127 也就是 1000 0000 到 0111 1111。-128 的绝对值二进制为 1000 0000，取反后 0111 1111。加 1 后是 1000 0000。-128 没有原码和反码，+0 和-0 的补码都是 0000 0000。

4.==运算符

答案：==的用法，它属于比较运算符，结果是 boolean 类型的，如果是基本数据类型，比较的是==两边的值，如果是引用数据类型，比较的是引用变量指向的那个地址值。另外对于字符串要用双引号引起来，如果是比较字符串的内容是否相同就用 equals 方法

5.Switch 问题

答案：switch 语句在 JDK1.5 之前只支持四种数值的判断，分别为：byte，short，int 和 char，JDK1.5 开始支持对枚举类型变量的判断，JDK1.7 开始支持对 String 类型变量的判断。switch 中 case 的常量表达式要和 switch 后面的表达式类型一致

6.什么是实例变量？什么是类变量

答案：①类变量也叫静态变量，也就是在变量前加了 static 的变量，类变量在创建对象前就已经在内存中存在，随类的创建而创建；实例变量也叫对象变量，即没加 static 的变量。

②类变量是所有对象共有，其中一个对象将它值改变，其他对象得到的就是改变后的结果；

而实例变量则属对象私有，某一个对象将其值改变，不影响其他对象。

③所有的实例对象都共用一个类变量，内存中只有一处空间是放这个类变量值的。因此，如果一个对象把类变量值改了，另外一个对象再取类变量值就是改过之后的了。在创建实例对象的时候，内存中会为每一个实例对象的每一个非静态成员变量开辟一段内存空间，用来存储这个对象所有的非静态成员变量值，即使两个不同的实例对象是属于同一个 class 类，但是它们的同名非静态成员变量在内存中占用的空间是不同的。

例子：public class A{

```
    static   int a = 0; //类变量
    private int b = 0; //实例变量
}
```

```
Public class B{
    public void main (String[] args){
        A a1 = new A();
        A a2 = new A();
        a1.a = 3;  // 等同于 A.a = 3;
        a1.b = 4 ;
        System.out.println(a2.a); //结果为 3
        //类变量是针对所有对象的，所以 a1 改变 a，a2 的 a 也改变
        System.out.println(a2.b); //结果为 0
        //实例只改变自身的，所以 a1 对象的 b 改变，不影响对象 a2 的 b 变量
    }
}
```

7.三元运算

答案：①int x=8,y;

y=(x>5)?100：200;

代码 y 参与运算，而没有初始化变量 y 的值，会报错。只需要在初始化 y=0 即可。

②int x=8,y;

(x>5)?y=100：y=200;

代码 (x>5)，先进行比较运算，然后给 y 赋值。

③要点：变量初始化的时候需要赋值才能参与运算。

8.两种创建数组的内存情况

答案：

```
class Practise  {
    public static void main(String[] args)  {
        int [] arr = new int[]{3,1,6,8,2,5};
        int [] arr1 ={3,1,6,8,2,5};
        System.out.println(arr.equals(arr1));//结果 false

        String[] aa=new String[]{"aa","bb","cc"};
        String[] bb={"aa","bb","cc"};
        System.out.println(aa.equals(bb));//结果 false

        String a="abc";
        String b=new String("abc");
        String c=new String("abc");
        System.out.println(a==b);//结果 false
        System.out.println(a.equals(b));//结果 true
        System.out.println(b.equals(c));//结果 true
        System.out.println(a.equals(c));//结果 true
```

```
    }
  }
```

实验证明，String 类是一个特殊的类，String b=new String("abc");在堆内存中重新开辟一块空间；虚拟机先查找有没有"abc"这个实体，有，则 a 直接指向实体，没有，则重新创建实体。而其他类即使是字符类型的数组也没有区别，仅仅是写法上的不同。

9.String 创建对象的内存问题

答案：

```
class StringDemo {
        public static void main(String[] args)  {
            String s1="abc";
            String s2=new String("abc");
            System.out.println(s1==s2);
/*这个比较的是 s1 和 s2 的内存地址值，它们两个分别在内存里面各自对应一个内存地址，所以它们相比的结果是 False*/
            System.out.println(s1.equals(s2));
 /*这两个比较的是两个对象的字符串是否相同，因为 String 类对象覆写了 Object 类中的 equals 方法。所以它们相比的结果是 true;*/
        }
    }
```

10.For 和 while 的区别

答案：while 的执行体和跌代体在一起，如果执行体后面加上 continue 后面的跌代体执行不到；for 的执行体和跌代替分离，执行体后面加 continue 跌代体还是是会执行。for 里面除了两个分号不能省略之外其余都可以省略！

11.垃圾回收机制

答案：Java 中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

某个对象不在使用 Demo d = new Demo（）； 当 d 不用的时就让 d = null；如此一来就会让垃圾回收器来主动回收。

12.==与 equals 的区别

答案：==和 equals 都是比较的,而前者是运算符,后者则是一个方法,基本数据类型和引用数据类型都可以使用运算符==,而只有引用类型数据才可以使用 equals,下面具体介绍一下两者的用法以及区别.

==操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用==操作符。

如果一个变量指向的数据是对象类型的，那么，这时候涉及了两块内存，对象本身

占用一块内存（堆内存），变量也占用一块内存，例如 Objet obj = new Object();变量 obj 是一个内存，new Object()是另一个内存，此时，变量 obj 所对应的内存中存储的数值就是对象占用的那块内存的首地址。对于指向对象类型的变量，如果要比较两个变量是否指向同一个对象，即要看这两个变量所对应的内存中的数值是否相等，这时候就需要用==操作符进行比较。

equals 方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。例如，对于下面的代码：

```
String a=new String("heima");
String b=new String("heima");
```

两条 new 语句创建了两个对象，然后用 a,b 这两个变量分别指向了其中一个对象，这是两个不同的对象，它们的首地址是不同的，即 a 和 b 中存储的数值是不相同的，所以，表达式 a==b 将返回 false，而这两个对象中的内容是相同的，所以，表达式 a.equals(b)将返回 true。

equals 本身是一个方法,它是根类 Object 里边的方法,所有类和接口都直接或者间接继承自 Object,所以在所有的类中都有 equals()方法,都是继承来的

在实际开发中，我们经常要比较传递进行来的字符串内容是否等，例如，String input = …;input.equals(“quit”)，许多人稍不注意就使用==进行比较了，这是错误的，随便从网上找几个项目实战的教学视频看看，里面就有大量这样的错误。记住，字符串的比较基本上都是使用 equals 方法。

如果一个类没有自己定义 equals 方法，那么它将继承 Object 类的 equals 方法，Object 类的 equals 方法的实现代码如下：

```
boolean equals(Object o){
return this==o;
}
```

这说明，如果一个类没有自己定义 equals 方法，它默认的 equals 方法（从 Object 类继承的）就是使用==操作符，也是在比较两个变量指向的对象是否是同一对象，这时候使用 equals 和使用==会得到同样的结果，如果比较的是两个独立的对象则总返回 false。如果你编写的类希望能够比较该类创建的两个实例对象的内容是否相同，那么你必须覆盖 equals 方法，由你自己写代码来决定在什么情况即可认为两个对象的内容是相同的。

代码,试一下 equals 和==的区别

```
/*
*    ==和 equals 的用法以及区别
* */
public class TestEqulas{
    public static void main(String[] args) {
        String s = new String("heima");
        String s2 = new String("heima");
        System.out.println(s.equals(s2));//输出 true,因为 String 类已经重写了 equals
```

```
        System.out.println(s == s2);//输出 false,因为两者的地址是不同的
    //创建三个动物对象
    //a1 和 a2name 和 age 都相同
        Animal a1 = new Animal("Tom",5);
        Animal a2 = new Animal("Tom",5);
    //先试一下用==比较链各个对象
        System.out.println(a1 == a2);
//输出 false,两个对象内容相同,但是他们的引用首地址不同
// 首先将自己写的 equals 方法注释掉,看输出结果是什么
    boolean b = a1.equals(a2);
    System.out.println(b);//结果为 false,证明是调用的继承来的那个 equals 方法
// 然后我们调用自己已经重写的 equals,再看下结果
    b = a1.equals(a2);//现在调用的是已经重写后的方法
    System.out.println(b);//所以打印的是 true
  }
  private String name;
  private int age;
  public Animal(){}
  public Animal(String name,int age){
    this.name = name;
    this.age = age;
  }
  public void setName(String name){
    this.name = name;
  }
  public String getName(){
    return name;
  }
  public void setAge(int age){
    this.age = age;
  }
  public int getAge(){
    return age;
  }
  //重写了 equals 方法
  public boolean equals(Object o){
    //判断两个对象是否为同一个对象,如果是就直接返回 true
    if (this == o) {
```

```
            return true;
        }
// 再判断另一个对象 o 是否是 null,如果是 null 就没有必要再比了,肯定不相等,直接返回 false
        if (o == null) {
            return false;
        }
//  如果前两个要求都符合，那么再判断对象 o 是否为 Animal 的实例，如果对象 o 是一个 Student 对象也就直接返回 false
        if (o instanceof Animal) {
// 如果是当前类的实例，那么就强制转换成当前类的实例，再依次比较成员变量是否相等
            Animal s = (Animal)o;
//  注意：String 类型的成员变量也可以看做是一个 String 对象，需要用 equals 比较，而不能用==比较
            if (this.getName().equals(s.name) && this.age == s.age) {
                return true;
            }
        }
        return false;
    }
}
```

13.堆内存与栈内存的区别

答案：①heep （堆）是一个可动态申请的内存空间，一般所有创建的对象都放在这里。stack （栈）是一个先进后出的数据结构，通常用于保存方法(函数)中的参数,局部变量。stack （栈）的空间小，但速度比较快， 存放对象的引用，通过栈中的地址索引可以找到堆中的对象。

②栈（java stacks）也是线程私有的，它的生命周期与线程相同。虚拟机栈描述的是 java 方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

堆（java Heap）是 java 虚拟机所管理的内存中最大的一块。Java 堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。

14.变量不赋值与赋 null 的区别

答案：在内存中只有赋值了才初始化变量，全局变量自动赋予初始值，局部变量必须手动赋值， 最好全部都手动赋值，自动赋值赋的也是 0 或者 null。内存空间如果赋值 null ，就是进行了初始化。

15.可变参数为什么要定义在参数列表的后面

答案：void add(int a )
void add(int a, int b)
void add (int a,int b,int c)

比如这里的 add 方法就是向集合中添加元素的方法，我们发现这样的重载会使得代码很臃肿，而且复用性也不是很高。

在之前的 java 版本中是这样设计的：void add（int[] a），就是将接受到的参数类型变成一个数组，在方法中对数组进行遍历，这样数组中有几个元素，就添加几个元素，从而就简化了程序。后来出现了可变参数了，就可以将上边的代码写为：public void add(int ...is){}，虽然形式上发生了改变，但其内部的调用方式是没有变化的，也就是将接受来个数不确定的参数，装在一个数组中，而且从 int...开始后所有的参数，就会作为数组中的元素装进数组中。当然有的时候，参数中包含其他参数和可变参数，比如：public void add(String a,int ...is){}：那么从 int...开始之后的参数就是可变参数，而 a 这个参数就不装入到数组中了。那么，如果写成 public void add( int ...is , String a){} ：根据调用的规则，a 也会作为一个参数添加到数组中去，这当然与我们的程序设计初衷是不相符的。所以，在使用可变参数的时候，要将可变参数定义在参数列表的最后面。

16.绝对路径和相对路径的区别

答案： ①相对路径就是指由这个文件所在的路径引起的跟其它文件（或文件夹）的路径关系。使用相对路径可以为带来非常多的便利，为什么要安装 JDK 时候定义环境变量？就是因为要简化书写。可以指定默认的路径，这就跟导入 import 包是一样的，new 对象时候要把包名也写上，但是你 import 导入之后可以不用书写包名，相对路径也是一样的。

绝对路径就是指在平时使用计算机时要找到需要的文件就必须知道文件的位置 ,而表示文件的位置的方式就是路径。例如，只要看到这个路径："D：\图片\XXX.jpg"我们就知道"XXX.jpg"文件是在 D 盘的"图片"目录中。类似于这样完整的描述文件位置的路径就是绝对路径。

17.Final 关键字与宏的区别

答案：在 java 中一些数据的出现，值是固定的，可以通过取名字再加上 final 关键字修饰，可以增强程序的阅读性。如：final double PI=3.14;在 C++中也可以通过宏定义来实现这样的功能。如：#define PI 3.14。他们的区别从编译时的行为来看这个问题。C++中的宏在预处理阶段会被加进程序变成程序的一部分，就是复制过去。final 只是一个对象不可修改的属性或方法。

18.0.01+0.09

答案：public class MathTest{

```
        public static void main(String[]args){
            double a = 0.01, b = 0.09;
                System.out.print(a+b);
        }
}
```

打印的结果为什么是 0.09999999999999999 而非 0.1，发生精度损失，无论是 double

还是 float 都会遇到这个问题，就拿 float 说话，float 的 0.1 二进制形式是 00111011 10011001100110011001101,根据符号位换算为 10 进制表达的值精确应该是这样计算 11001100110011001101101 乘以 2 的负 27 次方，实际值是 0.100000001490116119384765625 这样就产生了实际误差。

示例： public float getLeftMoney() throws Exception {
// TODO Auto-generated method stub
float m = new MoneyDaoImpl().CountAllMoney();
float c = new DetailsDaoImpl().countDetailsMoney();
flaot less = m-c;
System.out.println(m);
System.out.println(c);
System.out.println(less);
return less;
}

解决方案：

```
public float getLeftMoney() throws Exception {
        // TODO Auto-generated method stub
        float m = new MoneyDaoImpl().CountAllMoney();
        float c = new DetailsDaoImpl().countDetailsMoney();
        BigDecimal b1 = new BigDecimal(Float.toString(m));
        BigDecimal b2 = new BigDecimal(Float.toString(c));
        System.out.println(m);
        System.out.println(c);
        Float less = b1.subtract(b2).floatValue();
        System.out.println(less);
        return less;
    }
```

19.类文件冲突

答案： /*需求：获取一段代码的运行时间
* 思路：利用 java 提供的类进行当前系统时间的调用，
* 利用结束时间减去开始时间，获得运行时间。
* System.currentTimeMillis();
* */

```
abstract class getTime{
    //利用构造函数进行当前时间的获取。
    public final void getTime() {
        long start=System.currentTimeMillis();
        runcode();
```

```
            long end=System.currentTimeMillis();
            System.out.println("the time =    "+(end-start));
        }
        public abstract void runcode();          //利用子类的覆盖，实现运行代码的修改
}
class gongzuo extends getTime{
        public void runcode(){
            for(int x=0;x<100;x++)
                System.out.print(x);
        }
}
public class GetTime {
        public static void main(String[] args){
            gongzuo a=new gongzuo();
            a.getTime();
        }
}
```

错误提示：java 中的类文件冲突：A resource exist with a different case

原因：windows 系统认为 GetTime.class 和 getTime.class 是一个文件。

20.基本数据类型强制转换

答案：基本类型的强制转换主要是包括隐式强转，和显示强转，隐式强转是自动转换，显示强转是手动转换，一般显示强转是因为类型自动隐式强转，如果不手动强转是会出现错误。

①byte b=4；

b=b+1；//错误，因为右侧变成了 int 型 ， 1 是 int 型，占四个八位，b 是 byte 型，占 1 个八位，他们相加，需要 4 个八位空间才能容纳下其和，所以，右侧隐式强转为 int 型，然而左侧是 byte 型，装不下，所以会报错。

②b=(char)(b+1);//正确，这个就是显示强转，我们手动把本来是 int 类型的类型转化为和左侧相同的 char 型已规避错误；

③b+=1；//正确，因为这个提前要进行一次判断，就是 1 是不是在 byte 范围内，结果是，所以就把 1 当做 byte 型，赋给左边。

④byte a=3，b=4，c;

c=a+b ;//错误 ,两个 byte 型相加 ,不应该还是 byte 类型？这是因为在 java 中 ,两个 byte 相加，为了防止再放到 byte 中溢出，所以将其强转为 int 型，这里的强转是隐式的，然而左边是 byte 类型数据，所以会报错。

⑤c=3+4；//正确， 原因是，右侧的和是 7，然后他就会判断，7 是不是在 byte 数据类型的范围内，结果是，所以就把 7 当做是 byte 类型，直接赋值给左边。所以正确。

⑥int a;

byte b=3,e;

short s=3,t;
e=s+b;//错误
t=s+b;//错误
a=s+b;//正确 这个例子得出一个结论就是，byte 类型数据，和别的类型的数据也是会转化为 int 类型。为什么一个 byte 类型，和一个 short 类型会转化为 int 类型，一个占 1 个八位，一个占 4 个八位，不是应该相加后转化为 short 类型？ 原因是当两个数相加后防止溢出，所以就转为 int，而 short 就放不下。
⑦byte b=3,e;
float f=3f;
double d=3;
f=b+f;//正确。
d=b+d;//正确。这两个和 byte 和 int 相加一样，都是隐式强转为所占空间更大的哪一个数据类型。

所以可以得出一个结论就是 byte 和别的数据类型的数据相加，如果另一个数据的数据类型<int 类型，也就是 short ，byte 小于 4 个八位的数，那么他就会隐式强转为 int 类型，而和别的相加就会隐式转化为更高位的数据类型。

21.创建对象时，内存问题

答案：class test{}
```
main{
    test t = new test();
}
```
①开辟了栈内存空间，有一个变量 t，存放堆内存地址 new test();
②class test{}; 中包含一个空的构造方法，以及其从 Object 类中继承的所有东西，会分配内存。如果对 new 出来的东西一直没有释放掉对它的引用，java 的垃圾收集机制无法对其进行回收的，当创建的对象足够多时，会内存溢出。

22.Null 属于什么类型

答案：①Java 中 4 个系统定义的常量：NaN 非数值、Inf 无穷大、-Inf 负无穷大、null 空。
②null 用来标识一个不确定的对象，可以赋给引用型变量，不可以赋给基本类型变量
③Object 是已知的存在所有类的超类,但不包含不存在的类,也不包含 null。基本数据类型不是类不包含在 Object 中。

## 三、Java 三大特性：

1.多态到底是什么

答案：多态是一种运行期的行为，不是编译期的行为。

多态：父类型的引用可以指向子类型的对象。

比如 Parent p = new Child();当使用多态方式调用方法时，首先检查父类中是否有该方法，如果没有，则编译错误；如果有，再去调用子类的该同名方法。如果想要调用子类中有而父类中没有的方法，需要进行强制类型转换，如上面的例子中，将 p 转换为子类 Child 类型的引用。因为当用父类的引用指向子类的对象，用父类引用调用方法时，找不到父类中

不存在的方法。这时候需要进行向下的类型转换，将父类引用转换为子类引用。

2.关于继承与实现的问题

答案：子类可以单继承父类并同时实现多个接口；接口可以多继承接口；

3.抽象关键字为什么不能和 private、static、final 共存

答案：①private 是私有的意思,当它修饰方法的时候子类是不能够继承父类私有方法的,但是 abstract 修饰的方法必须要被子类继承并且实现,所有两者冲突。static 是静态的意思,所谓静态就是被共享,而当它修饰方法的时候就是静态方法,静态方法是不用创建对象就可以调用的,当在有继承关系的时候,abstract 修饰的是抽象,不可以被实例化,抽象方法必须由子类去实现,就又和 static 冲突了。final 是最终的意思,也就是不能被继承或者重写,而 abstract 修饰的方法必须要由子类重写,所以也是冲突的。所以 abstract 只能和 public 以及 protected 一起使用。

②Static :A )静态变量 ,被 static 修饰的变量类似一个全局变量( Java 中没有此概念 )。当这个类被虚拟机第一次加载的时候，就会为该变量分配了内存空间。当该类创建实例时，并不会生成对 static 变量的拷贝。而是多个该类的实例共享使用该变量。所有该类的对象都可以操作这块存储空间。如果用 final 修饰就另当别论了。创建完成就需要进行初始化，定义时可以直接初始化。 如果需要通过计算来初始化你的 static 变量，可以声明一个 static 块，Static 块仅在该类被加载时执行一次，且在类被第一次装载时。

注意： static 定义的变量初始化会优先于任何其它非 static 变量，不论顺序如何。在涉及到继承的时候 ,会先初始化父类的 static 变量 ,然后是子类的 ,依次类推。可以使用”类名.变量名 “直接使用 ,并且被该类所有实例化对象共享。可以被类中所有方法使用( static 与非 static )。该类中某一个对象修改了变量的值，其他所有该类对象中的对应值都会随之改变。定义时初始化，或者通过静态代码块初始化

B )静态方法 ,被 static 修饰的方法我们称之为类方法。可以死通过类直接调用该方法，而没必要创建该类的实例后调用该方法。

注意：可以使用”类名.方法名 “直接使用。只能调用其他 Static 方法。只能使用 static 成员变量。不能以任何形式引用 this 和 super

用途：静态方法常常为应用程序中的其它类提供一些实用工具，在 Java 的类库中大量的静态方法正是出于此目的而定义的。Arrays 和 Collections

C ) 静态类，通常一个普通类不允许声明为静态的，只有一个内部类才可以。这时这个声明为静态的内部类可以直接作为一个普通类来使用，而不需实例一个外部类。

补充：static 表示 “全局” 或者 “静态” 的意思，用来修饰成员变量和成员方法，也可以形成静态 static 代码块，但是 Java 语言中没有全局变量的概念。

被 static 修饰的成员变量和成员方法独立于该类的任何对象。它不依赖类特定的实例，被类的所有实例共享。只要这个类被加载，Java 虚拟机就能根据类名在运行时数据区的方法区内定找到他们。因此，static 对象可以在它的任何对象创建之前访问，无需引用任何对象。

用 public 修饰的 static 成员变量和成员方法本质是全局变量和全局方法，当声明它类的对象时，不生成 static 变量的副本，而是类的所有实例共享同一个 static 变量。

static 变量前可以有 private 修饰，表示这个变量可以在类的静态代码块中，或者类的其他静态成员方法中使用（当然也可以在非静态成员方法中使用--废话），但是不能在其他类中通过类名来直接引用，这一点很重要。实际上你需要搞明白，private 是访问权限限定，static 表示不要实例化就可以使用，这样就容易理解多了。static 前面加上其它访问权限关键字的效果也以此类推。

static 修饰的成员变量和成员方法习惯上称为静态变量和静态方法，可以直接通过类名来访问，访问语法为： 类名.静态方法名(参数列表...)；类名.静态变量名 。用 static 修饰的代码块表示静态代码块，当 Java 虚拟机（JVM）加载类时，就会执行该代码块。

static 变量 ：按照是否静态的对类成员变量进行分类可分两种：一种是被 static 修饰的变量，叫静态变量或类变量；另一种是没有被 static 修饰的变量，叫实例变量。两者的区别是： 对于静态变量在内存中只有一个拷贝（节省内存），JVM 只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。 对于实例变量，没创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响（灵活）。

static 方法：静态方法可以直接通过类名调用，任何的实例也都可以调用，因此静态方法中不能用 this 和 super 关键字，不能直接访问所属类的实例变量和实例方法（就是不带 static 的成员变量和成员成员方法

③final

特点：A）用 final 修饰的变量表示常量，只能被赋一次值，不能修改。 final 修饰的基本类型变量：值不能被修改。final 修饰的引用类型变量（对象）：对象地址不能被修改，对象内部的成员可以被修改。被定义为 final 的对象引用只能指向唯一一个对象，不可以将它再指向其他对象，但是一个对象内部的值却是可以改变的。被 final 修饰的变量是一个常量，必须被赋值后才能使用。可以在定义时赋值，也可在构造方法中赋值。（只要在构造方法结束前给赋值就可以。）

B）用 final 修饰的方法不能被子类的方法覆盖；

C）用 final 修饰的类不能被继承，没有子类。final 类不能被继承，因此 final 类的成员方法没有机会被覆盖，默认都是 final 的。但是 final 类中的成员变量可以被定义为 final 或非 final 形式。在设计类时，如果类不需要有子类，类的实现细节不允许改变，那么就设计为 final 类。

D） final 不能用来修饰构造方法。

④static 和 final：static、 final 用来修饰成员变量和成员方法。对于变量，表示一旦给值就不可修改，并且通过类名可以访问。对于方法，表示不可覆盖，并且可以通过类名直接访问。

4.抽象类和接口的区别

答案：A）抽象类：含有 abstract 修饰符的 class 即为抽象类，abstract 类不能创建的实例对象。含有 abstract 方法的类必须定义为 abstract class，abstract class 类中的方法不必是抽象的。abstract class 类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那

么子类也必须定义为 abstract 类型。

B）接口：可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为 public abstract 类型 ,接口中的成员变量类型默认为 public static final。

C）区别：

a）抽象类可以有构造方法，接口中不能有构造方法。

b）抽象类中可以有普通成员变量，接口中没有普通成员变量

c）抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

d） 抽象类中的抽象方法的访问类型可以是 public，protected，但接口中的抽象方法只能是 public 类型的，并且默认即为 public abstract 类型。

e） 抽象类中可以包含静态方法，接口中不能包含静态方法

f）抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类型。

g）一个类可以实现多个接口，但只能继承一个抽象类。

5.重载和重写的区别

答案：①重载 Overload 表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

条件：A）方法名必须相同（大小写必须一致才算是相同）B）参数列表不同，参数列表不同又分为 :参数列表的个数不同 ;参数列表的排列顺序不同 ;参数列表的数据类型不同。必须同时满足条件 A 和 B 才叫方法的重载。

注意： A）在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是 fun(int,float)，但是不能为 fun(int,int)）；B）不能通过访问权限、返回类型、抛出的异常进行重载；C）方法的异常类型和数目不会对重载造成影响；D）对于继承来说，如果某一方法在父类中是访问权限是 priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

②重写 Override 表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

在覆盖要注意以下的几点：A）覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；B）覆盖的方法的返回值必须和被覆盖的方法的返回一致；C）覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；D）被覆盖的方法不能为 private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

6.匿名内部类中对接口的方法为什么要加 public

答案：对于接口当中常见的成员：而且这些成员都有固定的修饰符。①全局常量。public static final ②抽象方法。public abstract。由此得出结论，接口中的成员都是公共的权限，都是 public。

7.静态内部类的作用

答案：①静态内部类，就是定义在外部类的成员位置上，之所以用静态修饰，就是因为内部类中都是共享数据，没有特有数据，定义成静态的，就可以直接用类名访问，不用在堆内存中创建对象。比较节省空间。

②在进行代码程序测试的时候，如果在每一个 Java 源文件中都设置一个主方法(主方法是某个应用程序的入口，必须具有)，会出现很多额外的代码。而且最主要的是这段主程序的代码对于 Java 文件来说，只是一个形式，其本身并不需要这种主方法。但是该主方法又必不可少。在这种情况下，就可以将主方法写入到静态内部类中，从而不用为每个 Java 源文件都设置一个类似的主方法。这对于代码测试是非常有用，在一些中大型的应用程序开发中，是一个常用的技术手段。

8.构造代码块和构造函数的区别

答案： ①作用： 构造函数是给特定的对象进行初始化的； 构造代码块是对所有对象初始化。

②格式： 构造函数是： 类名{}。构造代码块： {}

③执行顺序： 静态代码块------构造代码块---------->构造函数

9.匿名内部类

答案：匿名内部类：没有名字的内部类是一个匿名子类对象。定义前提：内部类必须继承一个类或者实现接口。定义的格式：new 父类名&接口名(){ 定义子类成员或者覆盖父类方法 }.方法。使用场景：当函数的参数是接口类型引用时，如果接口中的方法不超过 3 个。可以通过匿名内部类来完成参数的传递。在创建匿名内部类时，该类中的封装的方法不要过多，最好两个或者两个以内。

```
①new Object(){
        void show(){
            System.out.println("show run");
        }
  }.show();
②Object obj = new Object(){
        void show(){
            System.out.println("show run");
        }
  };
  obj.show();
```

区别：第一个可是编译通过，并运行。

第二个编译失败，因为匿名内部类是一个子类对象，当用 Object 的 obj 引用

指向时，就被提升为了 Object 类型，而编译时检查 Object 类中是否有 show 方法，所以编译失败。

10.This 与 this（）的区别

答案：this(有参数/无参数) 用于调用本类相应的构造函数；this. 后跟方法或属性 指示本类的方法或属性。

super(有参数/无参数) 用于调用父类相应的构造函数 ;super. 后跟方法或属性(父类中指明的 public）

11.两种单例模式的区别

答案：饿汉式：先初始化对象；类一加载就存在；Single 类一进内存，就已经创建好对象。

懒汉式：对象是方法被调用时才初始化，也叫做对象的延时加载；类一加载就为 null（调用方法的时候才在内存中建立对象）；Single 类进内存，对象还没有存在，只有调用方法时才建立对象。

12.继承中的构造方法

答案：在继承中，子类的构造函数都会默认访问父类的构造函数的。在子类的每一个构造函数中的第一行，都有一个隐士的语句 super()，来调用父类与之参数相同的构造函数。如果父类的构造函数被手动指定，就必须在子类的构造函数的第一行中声明，去找设置的父类构造函数，否则编译无法通过。

13.Java 接口和 C++的虚类区别

答案：c++虚类相当与 java 里面的抽象类，与接口的不同之处如下：①一个子类只能继承一个抽象类（虚类），但能实现多个接口；②一个抽象类可以有构造方法，接口没有构造方法；③一个抽象类中的方法不一定是抽象方法，即其中的方法可以有实现（有方法体），接口中的方法都是抽象方法 ,不能有方法体 ,只有声明 ;④一个抽象类可以是 public、private、protected、default, 接口只有 public;⑤一个抽象类中的方法可以是 public、private、protected、default，接口中的方法只能是 public 和 default.

相同之处：都不能实例化。

14.类初始化顺序

答案：class Fu {

```
int num = 1;//最先 num 默认初始化=0；当调用 Fu 类构造函数在赋值为=1；
//Fu 类构造代码块；
{
        show();//被 Zi 类覆盖，调用 Zi 类 show()方法；
        System.out.println("fu 构造代码块！" + num);
}
Fu() {
        super();
        //Fu 类 num 显示初始化=1；
        //Fu 类构造代码块；
        System.out.println("fu 构造方法！" + num);
```

```
	}
	void show() {
		System.out.println("fu show 方法！" + num);
	}
}
class Zi   extends Fu{
	int num = 2;//最先 num 默认初始化=0；当调用 Zi 类构造函数在赋值为=2；
	//Zi 类构造代码块；
	{
		System.out.println("zi 构造代码块！" + num);
	}
	Zi() {
		super();
		//Zi 类 num 显示初始化=2；
		//Zi 类构造代码块；
		System.out.println("zi 构造方法！" + num);
	}
	void show() {
		System.out.println("zi show 方法！" + num);
	}
}
public class Test9 {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		new Zi();
	}
}
/*
 zi show 方法！0        //此时 Zi 类 num 还没有被赋值为=2；
 fu 构造代码块！1
 fu 构造方法！1
 zi 构造代码块！2
 zi 构造方法！2
 */
```

15.普通代码块、静态代码块、构造代码块区别

答案： ①普通代码块：在方法或语句中出现的{}就称为普通代码块。普通代码块和一般的语句执行顺序由他们在代码中出现的次序决定--“先出现先执行”

```
  public class CodeBlock01{
```

```
        public static void main(String[] args){
            {
                int x=3;
               System.out.println("1,普通代码块内的变量 x="+x);
            }
            int x=1;
            System.out.println("主方法内的变量 x="+x);
            {
                int y=7;
                System.out.println("2,普通代码块内的变量 y="+y);
            }
        }
    }
    /*
    运行结果：
   1,普通代码块内的变量 x=3
        主方法内的变量 x=1
         2,普通代码块内的变量 y=7
    */
```

②构造代码块：直接在类中定义且没有加 static 关键字的代码块称为{}构造代码块。构造代码块在创建对象时被调用，每次创建对象都会被调用，并且构造代码块的执行次序优先于类构造函数。

```
public class CodeBlock02{
    {
      System.out.println("第一代码块");
    }
    public CodeBlock02(){
        System.out.println("构造方法");
        }
        {
          System.out.println("第二构造块");
        }
      public static void main(String[] args){
          new CodeBlock02();
          new CodeBlock02();
          new CodeBlock02();
    }
}
```

```
/*
*
执行结果：
第一代码块
第二构造块
构造方法
第一代码块
第二构造块
构造方法
第一代码块
第二构造块
构造方法
*/
```

③ 静态代码块：在 java 中使用 static 关键字声明的代码块。静态块用于初始化类，为类的属性初始化。每个静态代码块只会执行一次。由于 JVM 在加载类时会执行静态代码块，所以静态代码块先于主方法执行。

```
//如果类中包含多个静态代码块，那么将按照"先定义的代码先执行，后定义的代码后执行"。
//注意：1 静态代码块不能存在于任何方法体内。2 静态代码块不能直接访问静态实例变量和实例方法，需要通过类的实例对象来访问。
class Code{
    {
      System.out.println("Code 的构造块");
    }
    static{
        System.out.println("Code 的静态代码块");
        }
    public Code(){
        System.out.println("Code 的构造方法");
        }
    }
public class CodeBlock03{
     {
      System.out.println("CodeBlock03 的构造块");
     }
     static{
        System.out.println("CodeBlock03 的静态代码块");
        }
        public CodeBlock03(){
```

```
        System.out.println("CodeBlock03 的构造方法");
    }
    public static void main(String[] args){
        System.out.println("CodeBlock03 的主方法");
        new Code();
        new Code();
        new CodeBlock03();
        new CodeBlock03();
    }
}
/*
CodeBlock03 的静态代码块
CodeBlock03 的主方法
Code 的静态代码块
Code 的构造块
Code 的构造方法
Code 的构造块
Code 的构造方法
CodeBlock03 的构造块
CodeBlock03 的构造方法
CodeBlock03 的构造块
CodeBlock03 的构造方法
*/
```

16.可以调用私有构造方法吗

答案：构造方法被 private 修饰符修饰后，构造方法可以被使用的范围就只能是在本类中被使用了，如果非要被类使用的话，那就只有内部类了。反射可以直接获取这个私有的构造方法。

17.子类实例化的初始化过程

答案：在继承关系初始化顺序如下：父类静态（静态代码块和静态成员：按代码书写先后顺序执行）-->子类静态（同父类）-->父类非静态（变量隐式初始化-->变量显示初始化）-->父类构造（构造代码块-->构造函数）->子类非静态（同父类）-->子类构造（同父类）。

## 四、异常：

1.Throw 和 throws 的区别

答案：throws 使用在函数上，后面跟的是异常类。可以跟多个，用逗号隔开。

throw 使用在函数内，后面跟的是异常对象。

2.Final,finally 和 finalize 的区别

答案：①**final**—修饰符（关键字）

如果一个类被声明为 final，意味着它不能再派生出新的子类，不能作为父类被继承。

因此一个类不能既被声明为 abstract 的，又被声明为 final 的。将变量或方法声明为 final，可以保证它们在使用中不被改变。被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为 final 的方法也同样只能使用，不能重载。

②**finally**—再异常处理时提供 finally 块来执行任何清除操作。

如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。

③**finalize**—方法名

Java 技术允许使用 finalize()方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize()方法以整理系统资源或者执行其他清理工作。finalize()方法是在垃圾收集器删除对象之前对这个对象调用的

3.Java 异常处理机制总结

答案：①对捕获到的异常对象进行常见方法操作：String getMessage()：获取异常信息。

②对多异常的处理：A）声明异常时，建议声明更为具体的异常。这样处理的可以更具体。B）对方声明几个异常，就对应有几个 catch 块。不要定义多余的 catch 块。如果多个 catch 块中的异常出现继承关系，父类异常 catch 块放在最下面。

③ Exceptoin 中有一个特殊的子类异常 RuntimeException 运行时异常。如果在函数内容抛出该异常，函数上可以不用声明，编译通过。如果在函数上声明了该异常，调用者可以不用进行处理，编译通过。

④自定义异常时：如果该异常的发生，无法在继续进行运算，就让自定义异常继承 RuntimeException。

4.Java 异常几种处理方式

答案：捕获异常,自己处理；throw 出去,让别人处理。

示例：

```
①public class A{
      try{
          异常语句...
}
catch(Exception e){
     e.getMessage();//自己处理
    }
}
②public class A throws Exception{
     异常语句...
}//throw 出去,让他人处理
③public class A{
      try{
         异常语句...
```

```
    }
catch(Exception e){
    e.getMessage();//自己处理
    throw new Exception ("");
    }
}
```

5.什么时候会用到自定义异常？

答案：当需要自己定义异常的名称和特有内容，增强阅读性、方便查找时。

6.Try-catch-finally 的执行顺序

答案：

```
public class Test {
        public static void main(String[] args) {
            System.out .println(test ());
        }
        public static String test() {
            try {
                System.out .println("try block");
                return test1 ();
            }
        finally {
                System.out .println("finally block");
               //return "finally";
            }
        }
        public static String test1() {
            System.out .println("return statement");
            return "after return";
        }
    }
```

结果：

```
try block
return statement
finally block
after return
```

分析：①try 语句块，return test1()，则调用 test1 方法。

②test1()执行后返回"after return"，返回值"after return"保存在一个临时区域里。

③执行 finally 语句块。若 finally 语句有返回值，则此返回值将替换掉临时区域的返回值。

④将临时区域的返回值送到上一级方法中。

7.Error 与 exception 的区别

答案：Exception：可以是可被控制(checked) 或不可控制的(unchecked) ；表示一个由程序员导致的错误 ；应该在应用程序级被处理。

Error：总是不可控制的(unchecked) ；经常用来用于表示系统错误或低层资源的错误 ；如何可能的话，应该在系统级被捕捉。

8.为什么父类不抛异常，子类就不能抛异常

答案：将父类引用传入给方法 A（没有抛异常），方法 A 使用父类引用。使用多态将子类对象传递给 A，如果子类抛出异常 B，因为 A 方法没有过处理异常 B，将会导致程序错误。

## 五、多线程：

1.继承 Thread 类，为什么要继承 run 方法

答案：Thread 实现了 Runnable 接口，而 run 方法是 Runnable 的方法， 接口中的方法默认 public abstract。如果继承 Thread 类，不重写 run 方法 ，不会报错 ，但是无法指定线程运行的代码。

2.进程和线程的区别

答案：进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的一个独立单位。

线程是进程的一个实体，是 CPU 调度和分派的基本单位，他是比进程更小的能独立运行的基本单位，线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源（如程序计数器，一组寄存器和栈），一个线程可以创建和撤销另一个线程；

进程和线程的关系：

①一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

②资源分配给进程，同一进程的所有线程共享该进程的所有资源。

③线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

④处理机分给线程，即真正在处理机上运行的是线程。

⑤线程是指进程内的一个执行单元，也是进程内的可调度实体。

线程与进程的区别：

①调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位。

②并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可以并发执行。

③拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源。

④系统开销：在创建或撤销进程的时候，由于系统都要为之分配和回收资源，导致系统的明显大于创建或撤销线程时的开销。但进程有独立的地址空间，进程崩溃后，在保护模式下不会对其他的进程产生影响，而线程只是一个进程中的不同的执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但是在进程切换时，耗费的资源较大，效率要差些。

线程的划分尺度小于进程，使得多线程程序的并发性高。另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大的提高了程序运行效率。

线程在执行过程中，每个独立的线程有一个程序运行的入口，顺序执行序列和程序的出

口。但是线程不能够独立执行，必须依存在应用程序中，有应用程序提供多个线程执行控制。从逻辑角度看，多线程的意义子啊与一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

3.出现死锁的条件

答案：出现死锁有 4 个必要条件：①互斥：存在互斥使用的资源，也就是临界资源；②占有等待：拥有资源的进程都在等待另外的资源；③非剥夺：进行所占有的资源是不可剥夺使用的资源；④循环等待：都在相互等待资源。

4.Synchronized 锁与 lock 锁有什么区别

答案：①synchronized 锁只锁括号里面的代码内容，一个方法或者一个类等等。如果被锁的代码抛出异常会自动释放锁资源。

②lock 锁锁定 lock 和 unlock 之间的代码，被锁的代码抛出异常不会自动释放锁资源，需要 try catch 后在 finally 里面手动 unlock 释放锁资源。

5.多线程中同步与锁

答案： Lock 替代了 synchronized；而 Condition 替代了 Object 中的监视器方法（wait,notify,notifyall）使用，Condition 可以通过 Lock 锁获取，一个 Lock 可以对应多个 Condition。

```
class Resource{
    private String name;
    private int count = 1;
    private boolean flag = false;//定义标志，用于进程间切换
    private Lock lock = new ReentrantLock();
    private Condition con = lock.newCondition();
//private Condition con_pro = lock.newCondition();
//private Condition con_con = lock.newCondition();
//此处抛了异常，在处理线程时一定要记得做异常处理
    public void set(String name)throws InterruptedException{
        lock.lock();//上锁，lock()与 unlock()其实就类似于 synchronized ，只是 lock 必须手动上锁和解锁
        try{
            while(flag)
            con.await();//线程等待，类似于 wait
//con_pro.await();
            this.name = name+"-->"+count++;
            System.out.println("生产者.."+this.name);
            flag = true;
            con.signalAll();//唤醒全部线程，类似于 notifyAll
//con_con.signal();
```

```
            }
            finally{
            lock.unlock();//释放锁
            }
        }
//此处抛了异常，在处理线程时一定要记得做异常处理
        public void out()throws InterruptedException{
            lock.lock();//上锁
            try{
                while(!flag)
                con.await();//线程等待
//con_con.await();
                System.out.println("消费者.."+this.name);
                flag = false;
                con.signalAll();//唤醒线程
//con_pro.signal();
            }
            finally{
                lock.unlock();//释放锁
            }
        }
}
//以上注释的代码，实现了等待本方线程和只唤醒对方线程的功能。
```

6.多线程中 stop 为什么不合适

答案 :多线程之间一般是有联系的 ,若用 stop 停止了线程 ,容易强行打断线程之间的联系，容易产生错误。

7.同步代码块和同步方法的区别

答案 :同步方法持有的锁匙是 this ,即本类对象。而同步代码块可以自定义一把锁 ,语法为：synchronized(对象){同步内容}，当然这个对象也可以是 this。同步代码块的作用域小于同步函数,而同步函数的作用域大于同步代码块，同步代码块效率相对更高。

8.Sleep 和 wait 的区别

答案：①sleep 是 Thread 类中的，wait 是 Object 中的；②sleep 会在指定时间之后自动唤醒，wait 需要其他线程调用 notify 或者 notifyAll 唤醒；③sleep 还有个最大的特点就是谁调用，谁睡觉，即使在 a 类中调用 b 的 sleep 方法，实际上还是 a 去睡觉。④wait 只能使用在同步控制方法或者同步控制块中使用，sleep 在任何地方都能被使用；⑤持有锁的线程执行 sleep，不释放锁，持有锁的线程执行到 wait（ ）时锁释放。

9.为什么没有同步效果

答案：void add(int n){

```
            Object obj=new Object();
            synchronized (obj) {
                同步代码
            }
        }
```

把 Object obj=new Object();定义在 add 方法内部，每次调用方法都会 new 一个对象，synchronized (obj) 判断的标记不是同一个锁，

可以直接用类的字节码文件加锁，

将 Object obj=new Object();

synchronized (obj)

改为：synchronized (TongBuHanShu.class)

10.线程的优先级

答案：线程优先级：1-10，代表线程被执行到概率的高低。当两个线程优先级相差不是很多的情况下，比如 4、5、6 执行概率不会有明显变化。如果线程优先级相差过多，例如 1、10.那么当优先级较高的程序过多时 ,优先级为 1 的线程可能永远执行不到。在编写程序时，不建议指定线程的优先级。

11.线程获取名称

答案：

```
class Test extends Thread{
        Test( String name){
                //调用了 Thread 有参数的构造方法。
                super(name);
        }
        public void run(){
                for(int i=0; i<60; i++){
                        //这里显示的名字是设置的名字。
                        System.out.println(this.getName()+"test..."+i);
                }
        }
}
public class ThreadTest {
        public static void main(String[] args) {
                //给每个对象设置名字。
                Test t1 = new Test("one");
                Test t2 = new Test("two");
                t1.start();
                t2.start();
                for(int i=0; i<60; i++){
```

```
                System.out.println("main..."+i);
            }
        }
}
```

在使用 Test 类中，使用 super 在显示线程名称时显示 one,two 的原因是：

①Test 继承了 Thread 类，在 Test 的构造方法中，调用了 super(name)方法，这个就是调用了父类的 public Thread(String name)构造方法 ,这个构造方法就是为线程命名的。

②如果在 Test 类的构造方法中不调用 super(name)，系统会默认给 Test 取名字，默认为 Thread-0 开始

③自己定义名字，可以使用 setName(String name)方式修改线程的名字。

## 12.线程的五个状态和特点

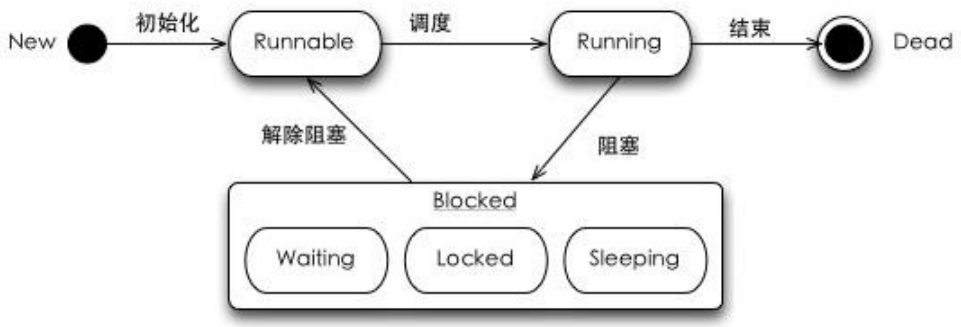
答案：①新建状态(New)：新创建了一个线程对象。

②就绪状态(Runnable)：线程对象创建后,其他线程调用了该对象的 start()方法.该状态的线程位于可运行线程池中,变得可运行,等待获取 CPU 的使用权.

③运行状态(Running)：就绪状态的线程获取了 CPU,执行程序代码.

④阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃 CPU 使用权,暂时停止运行.直到线程进入就绪状态,才有机会转到运行状态.阻塞的情况分三种：A）等待阻塞：运行的线程执行 wait()方法,JVM 会把该线程放入等待池中。B）同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用,则 JVM 会把该线程放入锁池中。 C）其他阻塞 :运行的线程执行 sleep()或 join()方法,或者发出了 I/O 请求时,JVM 会把该线程置为阻塞状态.当 sleep()状态超时 join()等待线程终止或者超时 或者 I/O 处理完毕时,线程重新转入就绪状态.

⑤死亡状态(Dead)：线程执行完了或者因异常退出了 run()方法,该线程结束生命周期.




## 13.什么情况下使用 Thread 什么情况下使用 Runnable

答案：继承 Thread 类：如果一个类有了父类，便无法再继承。实现 Runnale：更灵活，没有单继承的局限。

14.同步函数锁问题

答案：public class Demon {

//1.静态方法同步函数

public static synchronized void method1(){}

//2.非静态方法同步

public synchronized void method2(){}

//3.这个方法可以等价与静态方法同步函数 ,即静态方法同步函数省略了锁为 Demon.class 的静态代码块,因为静态方法是被类本身调用的 ,而每个类初始化的时候加载了 class 字节码，所以锁是 Demon.class

```
public void method3(){
    synchronized(Demon.class){}
}
```

//4 这个方法可以等价为非静态方法同步函数，也就是非静态方法底层的实现原理

//注意这个时候，静态代码块的锁就是 this,因为函数需要被对象调用，那么函数都有一个所属对象引用，所以用 this。

```
public void method4(){
    synchronized(this){}
}
```

//需要同步的一般是多线程的情况，并且涉及到操作共享数据。

}

## 六、常用 API

1.StringBuffer 与 StringBuilder 的区别

答案：① 在执行速度方面：StringBuilder > StringBuffer 。

②StringBuffer 与 StringBuilder，均为字符串变量，是可改变的对象，每当用它们对字符串做操作时，实际上是在一个对象上操作的，不像 String 一样创建一些对象进行操作，所以速度快。

③ StringBuilder：线程非安全的。StringBuffer：线程安全的

当在字符串缓冲去被多个线程使用是 ,JVM 不能保证 StringBuilder 的操作是安全的，虽然速度最快，但是可以保证 StringBuffer 是可以正确操作的。大多数情况下是在单线程下进行的操作，所以大多数情况下是建议用 StringBuilder 而不用 StringBuffer 的，就是速度的原因。

总结：如果要操作少量的数据用 = String；单线程操作字符串缓冲区 下操作大量数据 = StringBuilder；多线程操作字符串缓冲区 下操作大量数据 = StringBuffer。

2.Math 随机数问题

答案：①Math.random()生成的随机数范围默认在 0-1 之间的小数。

②new Random().nextInt() 生成的随机数是随机整数。

③Random 的对象来产生随机数，可以产生随机 int、float、double，long。

3.String s="a"+"b"+"c"内存创建了几个对象

答案：①String s = “abc” 和 String s = new String(“abc”) 的区别

String s = "abc" ;虚拟机首先会检查String池里有没有"abc"对象(通过equals方法)，如果有，直接返回引用，如果没有，会在池里创建一个“abc”对象，并返回引用。

```
String s1 = "abc";
String s2 = "abc";
System.out.println(s1==s2); // result: true
```

String str = new String("abc");不管缓冲池是否有"abc"，都会在堆内存创建一个"abc"对象，返回引用，此时，负责检查并维护缓冲池，其实堆内存的对象是缓冲池中"abc"对象的一个拷贝。

```
String s1 = new String("abc");
String s2 = new String("abc");
System.out.println(s1==s2); // result: false
```

③String s = “a” + “b” + “c” + “d” 创建了几个对象

String s = "a" + "b" + "c" + "d"; java 编译器有个合并已知量的优化功能，在编译阶段就把"a" + "b" + "c" + "d" 合并为 ” abcd “

```
String s = "a" + "b" + "c" + "d";
// String s = "abcd";
System.out.println(s=="abcd");// result: true
```

④String s1 = “a” String s2 = “b” String s3 = s1 + s2;

```
// String 是常量，不能相加的，java 的实现方式为：
StringBuilder sb = new StringBuidler(s1);
sb.append(s2);
s3 = sb.toString();
```

也就是说实际上 s3 是方法返回的 String 对象，凡是方法返回的字符串对象都是在堆内存的。

```
String s1 = "a";
String s2 = "b";
String s3 = s1 + s2; // 堆内存的对象
System.out.println(s3=="ab");// result: false
```

4.String 类为什么复写 Object 类的 equals 方法

答案：Object 类的 equals（ ）方法仅仅是比较两对象的 hashcode 地址值，而 String 类的 equalse()方法 比较的两个对象是否相等。

假设有一个 Person 类，并且 new 出了 p1 和 p2。p1 和 p2 内容相同，但是在堆中是不同的空间。此时使用 equals 方法得到的结果是 false，原因是两对象的引用地址不相同，使得比较失去实际意义，此时需要 String 类复写 Object 类的 equals 方法。

扩展：①之所以要去继承一个类，是因为该类的一些属性和方法是我们实际情况所需要的，所以我们不必再去定义另一个新的和它差不多的类，这样一来就减少了很多重复的代码，提高了代码的复用性。所以父类具有其所有子类的共性（属性和方法）。

②而方法的覆写是在继承的基础上的，没有继承就没有方法的覆写。之所以要覆写，是因为子类对父类的实现很不满意，或者说父类的实现没有达到子类的需求，实现的不够完整；所以子类只好把父类的方法给覆写了。补充：当子类去调用覆写后的方法时，调用的是子类覆写后的方法，而不是原来父类的方法。这就是动态绑定，即多态。

# 七、IO

1.字节流复制文件

答案：

```
package com.itheima;
import java.io.*;
//使用带缓冲功能的字节流复制文件
public class CopyFile{
	public static void main(String[] args){
		BufferedInputStream bis = null;
		BufferedOutputStream bos = null;
		try {
			FileInputStream fis = new FileInputStream("c:\\2.txt");
			bis = new BufferedInputStream(fis);
			FileOutputStream fos =
				new FileOutputStream("D:\\3.txt");
			bos = new BufferedOutputStream(fos);

			long len = bis.available();
			if(len > 1024*1024*200){
			    int b;
			    while((b=bis.read()) != -1){
			    bos.write(b);
			    }
			}
			else{
			byte[] bytes = new byte[(int)len];
			bis.read(bytes);
			bos.write(bytes);
			}
		System.out.println("文件复制已经完成！");
		}
		catch (Exception e) {
			e.printStackTrace();
		}
		finally{
```

```
                    if(bis != null){
                        try {
                            bis.close();
                        }
                        catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                    if(bos != null){
                        try {
                            bos.close();
                        }
                        catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }

        }
}
```

2.怎么删除带内容的文件夹

答案：

```
import java.io.*;
class removeDir {
        public static void main(String[] args) {
                File dir = new File("d:\\testDir");
                remove(dir);
        }
        public static void remove(File dir) {
                File[] files = dir.listFiles();
                for (int x = 0; x < files.length; x++) {
                        if (!files[x].isHidden() && files[x].isDirectory())
                                remove(files[x]);
                        else
                                System.out.println(files[x].toString() + "-file-"
                                                + files[x].delete());
                                // 删除的时候打印一下，以便看到是否重删等
                }
                System.out.println(dir.toString() + "-dir-" + dir.delete());
```

```
        }
}
```

4.FileWriter 默认缓冲区的大小

答案：无论使不使用 BufferedWriter，当向文件中写入数据时，数据流始终先保存到一段缓冲区中，当写入的数据流大于 8KB（使用 BufferedWriter 包装后是 16K）而没有进行刷新流操作时，JAVA 会自动刷新流，将流写入到文件中。

5.视频文件切割

答案：

```
import java.io.*;
public class Snippet {
        public static void main(String[]args)throws IOException{
                splitFile();
        }
        public static void splitFile() throws IOException{
                FileInputStream fis =
                        new FileInputStream("D:\\おおかみこどもの雨と雪.rmvb");
                FileOutputStream fos=null;
                byte buf[]=new byte[1024*1024*5];//定义一次写入 5M 数据
                int len=0;
                int count=1;
                fos=new FileOutputStream("d:\\"+(count++)+".part");
                int num = 1;//用来记录程序运行中实时的每个 output 写入次数
                        while ((len=fis.read(buf))!=-1){
                                fos.write(buf,0,len);
                                if(num>20) {
//如果写入次数大于 20(单个文件超过 100M),则此流停止,创建新的输出流写入新的片段
                                        fos.close();
                                        fos=
                                new FileOutputStream("d:\\"+(count++)+".part");
                                        num = 1;//重新计数
                                }
                                else{
                                        num++;
                                }
                        }
                        fis.close();
                        fos.close();
                        System.out.println("切割完毕!");
                }
```

```
}
```

6.批量更改文件名

答案：

```
package ReName;
import java.io.*;
class ReName{
	public static void main(String[] args) {
		String d = System.getProperty("user.dir");//获取当前文件路径；
		GL(d,".zip",20);
		//传递当前路径，资料后缀名，以及从多少位开始保留的文件名；
	}
	//过滤指定位置下指定类型的文件并更改文件名长度
	public static void GL(String dir,String hz,int len){
		File d = new File(dir);//建立 File 对象；
		String[] list = d.list(new FilenameFilter(){
		//后缀名过滤，并加入 list 合集中；
			public boolean accept(File dir,String name){
				return name.endsWith(hz);
			}
		});
		//for 循环控制批量重命名；
		for (int x = 0;x<list.length ;x++ ){
			String s1 = list[x];
			String s2 = s1.substring(len);//截取文件名长度；
			File f1 = new File(s1);
			File f2 = new File(s2);
			f1.renameTo(f2);//重命名；
		}
	}
}
```

7.字符输入流中 write（ ）方法

答案 :read()方法 :返回 :作为整数读取的字符 ,范围在 0 到 65535 之间 (0x00-0xffff)，如果已到达流的末尾，则返回 -1。

write(int c)方法：参数：c - 指定要写入字符的 int。

write 方法传入的是 int，打开被写入的文件变成字符，理由为（源码）：

```
public void write(int c) throws IOException {
	synchronized (lock) {
		if (writeBuffer == null){
			writeBuffer = new char[writeBufferSize];
```

```
                }
                writeBuffer[0] = (char) c;
                write(writeBuffer, 0, 1);
            }
        }
```

## 八、集合框架

1.List 和 Array 数组之间怎么互相转换

答案：数组转集合：asList()方法

```
String[] arr ={"123","456","789"};
List<String> list = Arrays.asList(arr);
```

集合转数组：toArray()方法

```
ArrayList<String> al = new ArrayList<String>();
al.add("abc1");
al.add("abc2");
al.add("abc3");
String[] arr = al.toArray(new String[al.size()]),
```

2.集合框架中容器简单用法

答案：①List 集合元素是有序，不唯一：

ArrayList 低层是数组，替代了 Vector，查询速度快；

LinkedList 低层是链表，，增删速度快；

Vector 低层也是数组，速度很慢，已经过时；

②Set 集合元素是无序，唯一；

HashSet 低层是哈希表；

LinkedHashSet 是 HashSet 的子类

TreeSet 底层是二叉树,

③Map 集合（键值对）

HashTable 低层是哈希表，不存在 null 键和 null 值；

HashMap 低层是哈希表，存在 null 键和 null 值；

TreeMap 低层是二叉树

技巧：Array 是数组结构，有角标，查询速度很快。link 就是链表结构：增删速度快，而且有特有方法，addFirst、 addLast、removeFirst()、removeLast()、getFirst()、getLast()。hash 就是哈希表，就要想要哈希值，就要想到唯一性，就要想到存入到该结构的中的元素必须覆盖 hashCode，equals 方法 tree 就是二叉树，就要想到排序，就想要用到比较。

4.Hashtable 和 hashMap 的区别

答案：① HashMap 不是线程安全的 ，是一个接口，是 map 接口的子接口，是将键映射到值的对象，其中键和值都是对象，并且不能包含重复键，但可以包含重复值。HashMap 允许 null key 和 null value，而 Hashtable 不允许。

②HashTable 是线程安全的一个 Collection。HashMap 是 Hashtable 的轻量级实现

（非线程安全的实现），他们都完成了 Map 接口，主要区别在于 HashMap 允许空（null）键值（key）,由于非线程安全，效率上可能高于 Hashtable。

③HashMap 允许将 null 作为一个 entry 的 key 或者 value，而 Hashtable 不允许。

④HashMap 把 Hashtable 的 contains 方法去掉了，改成 containsvalue 和 containsKey。因为 contains 方法容易让人引起误解。

⑤Hashtable 继承自 Dictionary 类，而 HashMap 是 Java1.2 引进的 Map interface 的一个实现。

⑥Hashtable 的方法是 Synchronize 的，而 HashMap 不是，在多个线程访问 Hashtable 时，不需要自己为它的方法实现同步，而 HashMap 就必须为之提供外同步。

⑦Hashtable 和 HashMap 采用的 hash/rehash 算法都大概一样，所以性能不会有很大的差异

5.Iterator 和 for 的区别

答案：①采用 ArrayList 对随机访问比较快，而 for 循环中的 get()方法，采用的即是随机访问的方法，因此在 ArrayList 里，for 循环较快。

采用 LinkedList 则是顺序访问比较快，iterator 中的 next()方法，采用的即是顺序访问的方法，因此在 LinkedList 里，使用 iterator 较快。

②for 循环适合访问顺序结构,可以根据下标快速获取指定元素.而 Iterator 适合访问链式结构,因为迭代器是通过 next()和 Pre()来定位的.可以访问没有顺序的集合。

③使用 Iterator 的好处在于可以使用相同方式去遍历集合中元素，而不用考虑集合类的内部实现（只要它实现了 java.lang.Iterable 接口），如果使用 Iterator 来遍历集合中元素，一旦不再使用 List 转而使用 Set 来组织数据，那遍历元素的代码不用做任何修改，如果使用 for 来遍历，那所有遍历此集合的算法都得做相应调整,因为 List 有序,Set 无序,结构不同,他们的访问算法也不一样。

6.LinkedList 为什么有索引还慢

答案：ArrayList：基于数组的遍历查找。LinkedList：基于链表的遍历查找。按照对象在内存中存储的顺序去考虑，数组的访问要比链接表快，因为对象都存储在一起。

## 九、网络编程

1.正则表达式获取字符串中 ip 地址

答案：

```
package com.itcast.regex;
import java.util.TreeSet;
public class IpRegexPractice {
	public static void main(String[] args) {
		ipSort("192.168.1.154	196.154.01.002	10.10.10.10
			2.2.2.2	198.40.5.4	187.255.255.255");
	}
	public static void ipSort(String ip){
		//都视为一位数字，前面先加上 00
		ip=ip.replaceAll("(\\d+)", "00$1");
```

```
                System.out.println(ip);
                //去掉数字前面多余的 0
                ip=ip.replaceAll("0*(\\d+{3})","$1");
                System.out.println(ip);
                //获取地址值
                String[]arr=ip.split(" +");
                TreeSet<String> ts=new TreeSet<String>();
                for (String string : arr) {
                        ts.add(string);
                }
                for (String string : ts) {
                        System.out.println(string.replaceAll("0*(\\d+)","$1"));
                }
        }
}
```

## 十、高新部分

1.反射手段创建类对象

答案：

```
class User{
        /*private User(){//将默认的构造方法私有化的话就不可以再创建对象，两种方法都是这样}*/
        public String toString() {
                return "User 对象创建成功！";
        }
}
public class NewInstanceDemo6 {
        public static void main(String[] args) throws Exception {
                //传统方式创建对象
                 System.out.println(new User());
                 //使用反射的方式
                 Class<User> c = User.class;
                 User u = c.newInstance();
                 /*直接 newInstance 的话必须保证默认的构造方法正常存在，也就是没有被私有化！这是前提条件*/
                 System.out.println(u);
        }
}
```

2.Invoke 方法

答案：invoke(Object obj, Object... args)中 Object... args 是多参数形式，就是说可以传递 0

到 n 个参数都是正确的，所以传递空也是可以的。

3.3 种获取字节码的方式什么时候用

答案：①类名.class，例如，System.class 通常用于获取已知类或者类型的.class 文件

②对象.getClass()，例如，new Date().getClass() 通常用于获取自定义对象的.class 文件方法所属类：

|--java.lang.Object

|--|--class<?> getClass() 返回 object 的运行时类

③Class.forName("类名")，例如，Class.forName("java.util.Date") 通常用于获取 API 文档中的.class 文件

方法所属类：

|--java.lang.Class

|--|--static Class<?> forName(String className) 返回与带有给定字符串名的类或接口相关联的 Class 对象。

5.动态代理原理

答案：当要给某些类的方法增加一些额外的功能时，我们可以通过创建另外一个代理类，在代理类中实现和另一个类的相同的这个方法，然后在代理类中增加额外的系统功能，当需要使用到系统功能时，调用该代理类的方法，也可以达到调用原来方法的目的，不需要系统功能时调用原来的类的方法。但是这样一来我们就要自己手动创建每个代理类，其实这也就是静态代理。Java 中的 Proxy 类可以动态创建代理类，为已经存在的具有相同接口的目标类的每个方法增加一些额外的系统功能，核心之处其实就是反射，Proxy 的 getProxyClass() 方法动态创建代理类，运行时 JVM 自动的生成代理类的.class 字节码文件。

```
public class ProxyTest {
    public static void main(String[] args) throws Exception {
        Collection proxy3 =
    (Collection)Proxy.newProxyInstance(Collection.class.getClassLoader(),
            new Class[] {Collection.class}, new InvocationHandler() {
            //创建目标类示例对象
                ArrayList target = new ArrayList();
                //重写 invoke()方法，每次调用代理类时，都计算目标方法的运行时间
                @Override
                public Object invoke(Object proxy, Method method,
                        Object[] args) throws Throwable {
                            long begin = System.currentTimeMillis();
                            Object retValue = method.invoke(target, args);
                            long end = System.currentTimeMillis();
                            System.out.println("++++" + (end - begin));
                            return retValue;
             }
```

```
            });
        proxy3.add("abc");
        System.out.println(proxy3);//打印目标集合
        System.out.println(proxy3.getClass().getName());//com.sun.proxy.$Proxy0
}
```

目标类是 ArrayList，Proxy3 对象可以当做目标类的对象来使用，它里面的方法有目标类的方法，实现目标类的功能就可以只接通过代理对象调用这些方法，就可以实现，同时也实现了系统功能(计算方法时间)。

6.javaBean 特点

答案：① JavaBean 是一个 public 的类

②JavaBean 有一个不带参数的构造方法

③ JavaBean 中的属性通过 getXXX()和 setXXX()进行操作

7.如何用反射得到数组类型

答案：

```
package test;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
public class Demo1 {
        /**
         * @param args
         * @throws IllegalAccessException
         * @throws InstantiationException
         * @throws SecurityException
         * @throws NoSuchMethodException
         * @throws InvocationTargetException
         * @throws IllegalArgumentException
         */
        public static void main(String[] args) throws InstantiationException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException,
NoSuchMethodException, SecurityException {
                // TODO Auto-generated method stub
                System.out.println(int[].class.getName());
                for(Field field:int[].class.getFields()){
                        System.out.println(field.getName());
                }
                for(Method method:int[].class.getMethods()){
                        System.out.println(method.getName());
```

```
                }
                for(Constructor constructor:int[].class.getConstructors()){
                        System.out.println(constructor.getName());
                }
        }
}
```

8.类加载器有什么用

答案：类加载器（class loader）用来加载 Java 类到 Java 虚拟机中。一般来说，Java 虚拟机使用 Java 类的方式如下：Java 源程序（.java 文件）在经过 Java 编译器编译之后就被转换成 Java 字节代码（.class 文件）。类加载器负责读取 Java 字节代码，并转换成 java.lang.Class 类的一个实例。每个这样的实例用来表示一个 Java 类。通过此实例的 newInstance()方法就可以创建出该类的一个对象。比如 Java 字节代码可能是通过工具动态生成的，也可能是通过网络下载的。基本上所有的类加载器都是 java.lang.ClassLoader 类的一个实例。

9.GetAttribute 和 getParameter 的区别

答案：①HttpServletRequest 类有 setAttribute()方法，而没有 setParameter()方法

②当两个 Web 组件之间为链接关系时，被链接的组件通过 getParameter()方法来获得请求参数，例如假定 welcome.jsp 和 authenticate.jsp 之间为链接关系，welcome.jsp 中有以下代码：

```
<a href="authenticate.jsp?username=wolf">authenticate.jsp </a>
```

或者：

```
<form name="form1" method="post" action="authenticate.jsp">
请输入用户姓名：<input type="text" name="username">
<input type="submit" name="Submit" value="提交">
</form>
```

在 authenticate.jsp 中通过 request.getParameter("username")方法来获得请求参数 username:

```
<% String username=request.getParameter("username"); %>
```

③当两个 Web 组件之间为转发关系时，转发目标组件通过 getAttribute()方法来和转发源组件共享 request 范围内的数据。

假定 authenticate.jsp 和 hello.jsp 之间为转发关系。authenticate.jsp 希望向 hello.jsp 传递当前的用户名字，如何传递这一数据呢？先在 authenticate.jsp 中调用 setAttribute()方法：

```
<%
String username=request.getParameter("username");
request.setAttribute("username"，username);
%>
<jsp:forward page="hello.jsp" />
```

在 hello.jsp 中通过 getAttribute()方法获得用户名字:

```
<% String username=(String)request.getAttribute("username"); %>
Hello: <%=username %>
```

从更深的层次考虑，request.getParameter()方法传递的数据，会从 Web 客户端传到 Web 服务器端，代表 HTTP 请求数据。request.getParameter()方法返回 String 类型的数据。

request.setAttribute()和 getAttribute()方法传递的数据只会存在于 Web 容器内部，在具有转发关系的 Web 组件之间共享。这两个方法能够设置 Object 类型的共享数据。request.getParameter()取得是通过容器的实现来取得通过类似 post，get 等方式传入的数据。

request.setAttribute()和 getAttribute()只是在 web 容器内部流转，仅仅是请求处理阶段。

getAttribute 是返回对象,getParameter 返回字符串。

总的来说：request.getAttribute()方法返回 request 范围内存在的对象，而 request.getParameter()方法是获取 http 提交过来的数据。

10.JVM 加载 class 文件的原理机制

答案：①Java 中的所有类，必须被装载到 jvm 中才能运行，这个装载工作是由 jvm 中的类装载器完成的,类装载器所做的工作实质是把类文件从硬盘读取到内存中

②java 中的类大致分为三种：

1.系统类

2.扩展类

3.由程序员自定义的类

③类装载方式，有两种

1.隐式装载， 程序在运行过程中当碰到通过 new 等方式生成对象时，隐式调用类装载器加载对应的类到 jvm 中，

2.显式装载， 通过 class.forname()等方法，显式加载需要的类

隐式加载与显式加载的区别： 两者本质是一样，

④类加载的动态性体现

一个应用程序总是由 n 多个类组成，Java 程序启动时，并不是一次把所有的类全部加载后再 运行，它总是先把保证程序运行的基础类一次性加载到 jvm 中，其它类等到 jvm 用到的时候再加载，这样的好处是节省了内存的开销，因为 java 最早就是为嵌入式系统而设计的，内存宝贵，这是一种可以理解的机制，而用到时再加载这也是 java 动态性的一种体现。

⑤java 类装载器

Java 中的类装载器实质上也是类，功能是把类载入 jvm 中，值得注意的是 jvm 的类装载器并不是一个，而是三个，层次结构如下：

```
Bootstrap Loader - 负责加载系统类
        |
    - - ExtClassLoader - 负责加载扩展类
```

|

- - AppClassLoader - 负责加载应用类

三个类加载器，一方面是分工，各自负责各自的区块，另一方面为了实现委托模型。

⑥类加载器之间是如何协调工作的

在这里 java 采用了委托模型机制，这个机制简单来讲，就是“类装载器有载入类的需求时，会先请示其 Parent 使用其搜索路径帮忙载入，如果 Parent 找不到,那么才由自己依照自己的搜索路径搜索类”，注意喔，这句话具有递归性

Java 代码 [url=][img][/img][/url]

```
/**
 * @author Jamson Huang
 *
 */
public class TestClass {

    /**
     * @param args
     */
    public static void main(String[] args)  throws Exception{
        //调用 class 加载器
        ClassLoader cl = TestClass.class.getClassLoader();
        System.out.println(cl);
        //调用上一层 Class 加载器
        ClassLoader clParent = cl.getParent();
        System.out.println(clParent);
        //调用根部 Class 加载器
        ClassLoader clRoot = clParent.getParent();
        System.out.println(clRoot);

    }

}
```

Result 代码 [url=][img][/img][/url]

Run， Console 中出现的 log 信息如下：

```
sun.misc.Launcher$AppClassLoader@7259da
sun.misc.Launcher$ExtClassLoader@16930e2
null
```

可以看出TestClass是由AppClassLoader加载器加载的 ,AppClassLoader的Parent加载器是 ExtClassLoader ,但是 ExtClassLoader 的 Parent 为 null,依 java 的观点来看，逻辑上并不存在 Bootstrap Loader 的类实体，所以在 java 程序代码里试图打印出其内容时，我们就会看到输出为 null

【注：以下内容大部分引用 java 深度历险】

弄明白了上面的示例，接下来直接进入类装载的委托模型实例，写两个文件，如下：

Java 代码 [url=][img][/img][/url]

```
/**
 * @author Jamson Huang
 *
 */
public class Test1 {

    /**
     * @param args
     */
    public static void main(String[] args)throws Exception {
        System.out.println(Test1.class.getClassLoader());

        Test2 test2 = new Test2();

        test2.print();
    }

}
/**
* @author Jamson Huang
*
*/
public class Test2 {
    public void print(){
        System.out.println(Test2.class);
        System.out.println(this.getClass());
        System.out.println(Test2.class.getClassLoader());
    }
}
```

Result 代码 [url=][img][/img][/url]

Run,Console 出现 log 如下：

1. sun.misc.Launcher$AppClassLoader@7259da
2. class com.java.test.Test2
3. class com.java.test.Test2
4. sun.misc.Launcher$AppClassLoader@7259da

⑦ 预先加载与依需求加载

Java 运行环境为了优化系统,提高程序的执行速度,在 JRE 运行的开始会将 Java 运行所需要的基本类采用预先加载( pre-loading )的方法全部加载要内存当中，因为这些单元在 Java 程序运行的过程当中经常要使用的，主要包括 JRE 的 rt.jar 文件里面所有的 .class 文件。

当 java.exe 虚拟机开始运行以后，它会找到安装在机器上的 JRE 环境，然后把控制权交给 JRE ， JRE 的类加载器会将 lib 目录下的 rt.jar 基础类别文件库加载进内存，这些文件是 Java 程序执行所必须的，所以系统在开始就将这些文件加载，避免以后的多次 IO 操作，从而提高程序执行效率。

图( 2 )我们可以看到多个基础类被加载， java.lang.Object,java.io.Serializable 等等。

相对于预先加载，我们在程序中需要使用自己定义的类的时候就要使用依需求加载方法( load-on-demand ),就是在 Java 程序需要用到的时候再加载,以减少内存的消耗，因为 Java 语言的设计初衷就是面向嵌入式领域的。

⑧自定义类加载机制

之前我们都是调用系统的类加载器来实现加载的,其实我们是可以自己定义类加载器的。利用 Java 提供的 java.net.URLClassLoader 类就可以实现。下面我们看一段范例：

Java 代码 [url=][img][/img][/url]

```
1. try{
2. URL url = new URL("file:/d:/test/lib/");
3. URLClassLoader urlCL = new URLClassLoader(new URL[]{url});
4. Class c = urlCL.loadClass("TestClassA");
5. TestClassA object = (TestClassA)c.newInstance();
6. object.method();
7. }catch(Exception e){
8. e.printStackTrace();
9. }
```

我们通过自定义的类加载器实现了 TestClassA 类的加载并调用 method ( )方法。分析一下这个程序：首先定义 URL 指定类加载器从何处加载类， URL 可以指向网际网络上的任何位置，也可以指向我们计算机里的文件系统( 包含 JAR 文件 )。上述范例当中我们从 file:/d:/test/lib/ 处寻找类；然后定义 URLClassLoader 来加载所需的类，最后即可使用该实例了。

⑨ 类加载器的阶层体系

当执行 java ***.class 的时候， java.exe 会帮助我们找到 JRE ，接着找到位于 JRE

内部的 jvm.dll ,这才是真正的 Java 虚拟机器，最后加载动态库 ,激活 Java 虚拟机器。虚拟机器激活以后，会先做一些初始化的动作，比如说读取系统参数等。一旦初始化动作完成之后，就会产生第一个类加载器¯ Bootstrap Loader ， Bootstrap Loader 是由 C++所撰写而成，这个 Bootstrap Loader 所做的初始工作中，除了一些基本的初始化动作之外，最重要的就是加载 Launcher.java 之中的 ExtClassLoader ，并设定其 Parent 为 null ，代表其父加载器为 BootstrapLoader 。然后 Bootstrap Loader 再要求加载 Launcher.java 之中的 AppClassLoader ，并设定其 Parent 为之前产生的 ExtClassLoader 实体。这两个加载器都是以静态类的形式存在的。这里要请大家注意的是，Launcher$ExtClassLoader.class 与 Launcher$AppClassLoader.class 都是由 Bootstrap Loader 所加载，所以 Parent 和由哪个类加载器加载没有关系。

下面的图形可以表示三者之间的关系：

BootstrapLoader <---(Extends)----AppClassLoader

<---(Extends)----ExtClassLoader

这三个加载器就构成我们的 Java 类加载体系。他们分别从以下的路径寻找程序所需要的类：

BootstrapLoader ： sun.boot.class.path

ExtClassLoader: java.ext.dirs

AppClassLoader: java.class.path

这三个系统参量可以通过 System.getProperty() 函数得到具体对应的路径。大家可以自己编程实现查看具体的路径。

11.泛型与 C++模板有什么区别

答案：泛型本质上是提供类型的"类型参数"，它们也被称为参数化类型（parameterized type）或参量多态。其实泛型思想并不是 Java 最先引入的，C++ 中的模板就是一个运用泛型的例子。

①java 中没有 template 的关键字，c++中有。

②Java 语言中的泛型不能接受基本类型作为类型参数¯它只能接受引用类型。这意味着可以定义 List<Integer>，但是不可以定义 List<int>。

③在 java 中，尖括号通常放在方法名前，而 c++则是放在方法名后，c++的方式容易产生歧义，例如 g(f<a,b>(c))，这个则有两种解释，一种是 f 的泛型调用，c 为参数，a，b 为泛型参数。另一种解释，则是，g 调用，两个 bool 类型的参数。

④在 C++ 模板中，编译器使用提供的类型参数来扩充模板，因此，为 List<A> 生成的 C++ 代码不同于为 List<B> 生成的代码 ,List<A> 和 List<B> 实际上是两个不同的类。而 Java 中的泛型则以不同的方式实现，编译器仅仅对这些类型参数进行擦除和替换。类型 ArrayList<Integer> 和 ArrayList<String> 的对象共享相同的类，并且只存在一个 ArrayList 类。因此在 c++中存在为每个模板的实例化产生不同的类型 ,这一现象被称为“模板代码膨胀”，而 java 则不存在这个问题的困扰。java 中虚拟机中没有泛型，只有基本类型和类类型 ,泛型会被擦除 ,一般会修改为 Object ,如果有限制 ,例如 T extends Comparable, 则会被修改为 Comparable。而在 C++中不能对模板参数的类型加以限制，如果程序员用

一个不适当的类型实例化一个模板，将会在模板代码中报告一个错误信息。

12.怎样通过反射获取@Test 注解类

答案：Annotation [] ants= method1.getAnnotations();

```
        //获取这个方法的所有注解
        然后便利打印就行
        //自定义注解
        @Retention(RetentionPolicy.RUNTIME)
        //这样才能在运行时获取到注解不然获取不到  默认是 class 中
        @interface Test{}
```

13.ArrayList 代理

答案：

```
public class Test2 {
// ArraiList 代理类的测试
public static void main(String[] ages) {
   MyArrayListInterface myArrayList =ArrayListProxy.getArrayListProxy();
   myArrayList.add("AAAAA");
   System.out.println(myArrayList.get(0));
}
// 定义一个 ArrayList 的代理类
static class ArrayListProxy {
   // 定义一个返回 ArrayListProxy 对象的方法
   public static MyArrayListInterface getArrayListProxy() {
    MyArrayListInterface myArrayListProxy = (MyArrayListInterface) Proxy
          .newProxyInstance(MyArrayListInterface.class.getClassLoader(),
          new Class[] { MyArrayListInterface.class },new InvocationHandler() {
          // 定义一个 MyArrayList 对象
          private List myArrayList = new MyArrayList();
          public Object invoke(Object proxy,
            Method method, Object[] args)
            throws Throwable {
           // 定义记住此时时间的变量
           long start = System.currentTimeMillis();
           // 执行 al 的对应方法
           Object objReturn = method.invoke(myArrayList, args);
           // 输出被调用方法执行的所需时间
           System.out.println(method.getName()+"方法运行时间:"+
               (System.currentTimeMillis()- start)+"毫秒");
           return objReturn;// 返回被调用方法执行结果
          }
```

```
            });
     return myArrayListProxy;
   }
 }
}
// 定义一个实现 MyArrayListInterface 的类 MyArrayList
class MyArrayList extends ArrayList implements MyArrayListInterface {}
// 定义一个接口,同时继承 ArrayList 类实现的全部接口
interface MyArrayListInterface<E> extends Serializable, Cloneable, Iterable<E>,
   Collection<E>, List<E>, RandomAccess {}
```

14.注解是什么

答案：注解（Annotation） 为我们在代码中添加信息提供了一种形式化的方法，常见的作用有以下几种：

①生成文档。这是最常见的，也是 java 最早提供的注解。常用的有@see @param @return 等；

②跟踪代码依赖性，实现替代配置文件功能。比较常见的是 spring 2.5 开始的基于注解配置。作用就是减少配置。现在的框架基本都使用了这种配置来减少配置文件的数量；

③在编译时进行格式检查。如@Override 放在方法前，如果你这个方法并不是覆盖了超类方法，则编译时就能检查出；

注释有 3 中基本类型

a. 标记注释 -- 没有变量，只有名称标识。例如 @annotation

b. 单一值注释 -- 在标记注释的基础上提供一段数据。如 @annotation(“data”)

c. 完整注释 -- 可以包括多个数据成员，每个数据成员由名称和值构成。

@annotation(val1="data1",val2="data2")

Java 中提供 3 个内置注释类型：

a. Override ，只能用于方法（ 不能用于类，包声明或者其他构造 ）

作用：可以保证编译时候 Override 函数的声明正确性

用法： @Override

public void fun(){..}

b.Deprecated 同样只能作用与方法

作用：对不应再使用的方法进行注解

用法： @Deprecated public void fun{...} // 它们说这个注释跟函数要同一行

c.SupressWarnings 可以注释一段代码

作用：关闭特定的警告信息，例如你在使用泛型的时候未指定类型

用法： @SupressWarnings(value={"unchecked"})

d：@Safe Varargs(JAVA7 新增)

15.Java 泛型擦除

答案： Java 泛型（Generic）的引入加强了参数类型的安全性，减少了类型的转换，但有一点需要注意：Java 的泛型在编译器有效，在运行期被删除，也就是说所有泛型参数类型在编译后都会被清除掉，看下面一个列子，代码如下：

```
1. public class Foo {
2.     public void listMethod(List<String> stringList){ }
3.     public void listMethod(List<Integer> intList) { }
4. }
```

代码很简单，看起来没什么问题，但是编译器却报出如下错误信息：

Method listMethod(List<String>) has the same erasure listMethod(List<E>) as another method in type Foo

此错误的意思是说 listMethod(List<String>) 方法在编译时擦除类型后的方法是 listMethod(List<E>)，它与另外一个方法重复，也就是方法签名重复。反编译之后的方法代码如下：

```
1. public void listMethod(List list) { }
```

从上面代码可以看出 Java 编译后的字节码中已经没有泛型的任何信息，在编译后所有的泛型类型都会做相应的转化，转化如下：

- List<String>、List<T> 擦除后的类型为 List。
- List<String>、List<T>[] 擦除后的类型为 List[]。
- List<? extends E>、List<? super E> 擦除后的类型为 List<E>。
- List<T extends Serialzable & Cloneable> 擦除后类型为 List<Serializable>。

Java 为什么这么处理呢？有以下两个原因：

1. 避免 JVM 的大换血。如果 JVM 将泛型类型延续到运行期，那么到运行期时 JVM 就需要进行大量的重构工作了，提高了运行期的效率。
2. 版本兼容。 在编译期擦除可以更好地支持原生类型（Raw Type）。

明白了 Java 泛型是类型擦除的，下面的问题就很好理解了：

（1） 泛型的 class 对象是相同的

每个类都有一个 class 属性，泛型化不会改变 class 属性的返回值，例如：

```
1. public static void main(String[] args) {
2.     List<String> ls = new ArrayList<String>();
3.     List<Integer> li = new ArrayList<Integer>();
4.     System.out.println(ls.getClass() == li.getClass());
5. }
```

代码返回值为 true，原因很简单，List<String> 和 List<Integer> 擦除后的类型都是 List。

（2） 泛型数组初始化时不能声明泛型类型

如下代码编译时通不过：

```
1. List<String>[] list = new List<String>[];
```

在这里可以声明一个带有泛型参数的数组，但是不能初始化该数组，因为执行了类

型擦除操作后，List<Object>[] 与 List<String>[] 就是同一回事了，编译器拒绝如此声明。

（3） instanceof 不允许存在泛型参数

以下代码不能通过编译，原因一样，泛型类型被擦除了。

```
1. List<String> list = new ArrayList<String>();
2. System.out.println(list instanceof List<String>);
```

## 十一、经典问题

1.金额转换问题

答案：

```
public class RenMingBi {
        /**
         * @param args add by zxx ,Nov 29, 2008
         */
        private static final char[] data = new char[]{
                        '零','壹','贰','叁','肆','伍','陆','柒','捌','玖'
                };
        private static final char[] units = new char[]{
                '元','拾','佰','仟','万','拾','佰','仟','亿'
        };
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                System.out.println(
                                convert(135689123));
        }
        public static String convert(int money)
        {
                StringBuffer sbf = new StringBuffer();
                int unit = 0;
                while(money!=0)
                {
                        sbf.insert(0,units[unit++]);
                        int number = money%10;
                        sbf.insert(0, data[number]);
                        money /= 10;
                }
                return sbf.toString();
        }
}
```

2.阶乘后连续 0 个数问题

答案：10 = 2 * 5。每一个 2 与一个 5 相乘，结果就增加一个零。所以求 n! 后面的连续

零的个数，其实就是求其中相乘的数含有因子每对因子 2 与 5 的个数。又因为从 1 到某个数，所含 2 的个数比 5 多，所以问题就可以进一步简化到求含有因子 5 的个数。

JAVA 实现代码如下：

```
1.  static int zeroCount ( int n) {
2.      int counter = 0;
3.      for( int i = 5,m; i <= n; i += 5) {
4.          m = i;
5.          while ( m % 5 == 0) {
6.              counter++;
7.              m /= 5;
8.          }
9.      }
10.     return counter;
11. }
```

进一步优化算法：

```
1.  static int zeroCount ( int n) {
2.      int counter = 0;
3.      while ( n >= 5) {
4.          n /= 5;
5.          counter += n;
6.      }
7.      return counter;
8.  }
```

3.质数问题

答案： public static void main(String[] args){

Random rd = new Random(); //定义一个随机变量

int i = rd.nextInt(1000); //获取个随机数

System.out.println(isNum(i)); //输出结果

}

public static String isNum(int i ){

String ret =i+"是质数"; //默认是质数

if(i==2)

return ret; //如果是 2 返回默认结果

for (int j = 2; j < i/2; j++) { //制订循环次数

if(i%j==0){

ret=i+"不是质数"; // 如果能够整除返回不是质数

break;

}

}

return ret; //返回结果

}

4.数字黑洞问题（尚未解决哦）

问题：有任意一个 5 位数，如：34256，把它的各位数字打乱，重新排列，就可得到一个最大的数：65432，还有一个最小的数 23456。求这两个数字的差，得：41976，然后把这个数字再次重复上述过程（如果不足 5 位，则前边补 0）。如此往复，数字会落入某个循环圈

（称为数字黑洞）。比如，刚才的数字会落入：[82962, 75933, 63954, 61974] 这个循环圈。

请编写程序，找到 5 位数所有可能的循环圈，并输出，每个循环圈占 1 行。其中 5 位数全都相同则循环圈为 [0]，这个可以不考虑。循环圈的输出格式仿照：[82962, 75933, 63954, 61974]，其中数字的先后顺序可以不考虑。

5.百鸡问题

问题：3 文钱可以买 1 只公鸡，2 文钱可以买一只母鸡 1 文钱可以买文钱可以买 3 只小鸡。用 100 文 钱买 100 只鸡，那么各有公鸡母鸡小鸡多少只？

答案：遍历的时候当公鸡 母鸡 小鸡的价格总和操作 100 文的时 后面就可以不再遍历了 例如当公鸡 a= 20 的时候 母鸡 b 肯定<=20 20*3(公鸡的价格总和)+20*2（母鸡的价格总和）+小鸡的价格总和 >=100,这个时候循环就可以终止了，但是循环条件里面是 for(int b = 0; b<=50; b++)，所以对于 20<b<=50 这个取值区间没有必要在进行遍历了。

```
for(int a = 0; a<=33; a++){//最多 33 只公鸡
    for(int b = 0; b<=50; b++){//最多 50 只母鸡
        int c = 100 - a - b; //c 只小鸡
        if((a*3 + b*2 + c/3 == 100) && c%3 == 0 ){
            System.out.println(a + "只公鸡，" + b + "只母鸡，" + c + "只小鸡");
        }
        if(a*3 + b*2 + c/3 > 100)
            break;
    }
}
```

6.1~1000 累乘后面有多少个零问题

答案：

```
public static void main(String[] args) {
        BigInteger bi = new BigInteger("1");//初始化值为 1
        for(int i=2;i<1001;i++){
            bi=bi.multiply(BigInteger.valueOf(i));//这里是做乘法运算
        }
        int N=0;
        String s=bi.toString();//结果转化为字符串
        for(int i=0;i<s.length();i++){
            if(s.charAt(i)=='0'){
                N++;//有一个 0 就加上 1
            }
        }
        System.out.println("1000！中共有 0 的个数为："+N);
    }
```

7.猴子分桃问题（尚未解决哦）

问题：海滩上有一堆桃子，五只猴子来分。第一只猴子把这堆桃子凭据分为五份，多了一个，

这只猴子把多的一个扔入海中，拿走了一份。第二只猴子把剩下的桃子又平均分成五份，又多了一个，它同样把多的一个扔入海中，拿走了一份，第三、第四、第五只猴子都是这样做的，问海滩上原来最少有多少个桃子？

8.蚂蚁爬木杆问题

问题：有一根 27 厘米的细木杆，在第 3 厘米、7 厘米、11 厘米、17 厘米、23 厘米这五个位置上各有一只蚂蚁。木杆很细，不能同时通过一只蚂蚁。开始 时，蚂蚁的头朝左还是朝右是任意的，它们只会朝前走或调头， 但不会后退。当任意两只蚂蚁碰头时，两只蚂蚁会同时调头朝反方向走。假设蚂蚁们每秒钟可以走一厘米的距离。编写程序，求所有蚂蚁都离开木杆 的最小时间和最大时间。

9.螺旋矩阵问题

问题： 输出 n=5 的螺旋方阵

```
 1  2  3  4  5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
```

答案：

```
public class Main {
    public static void main(String[] args) {
        // 设定参数
        final int N = 7;
        final int COUNT = 11;
        // 初始化
        int mat[][] = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                mat[i][j] = 0;
            }
        }
        {// 写入
            int i = 0;
            int j = 0;
            int count = COUNT;
            mat[0][0] = count++;
            while (count < N * N + COUNT) {
                while (j + 1 < N && mat[i][j + 1] == 0) {
                    mat[i][++j] = count++;
                }
```

```
                    while (i + 1 < N && mat[i + 1][j] == 0) {
                        mat[++i][j] = count++;
                    }
                    while (j - 1 >= 0 && mat[i][j - 1] == 0) {
                        mat[i][--j] = count++;
                    }
                    while (i - 1 >= 0 && mat[i - 1][j] == 0) {
                        mat[--i][j] = count++;
                    }
                }
            }
            //输出
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                    String out = String.valueOf(mat[i][j]);
                    String str = "     ";
                    out = str.substring(0, 5 - out.length()) + out;
                    System.out.print(out);
                }
                System.out.println();
            }
        }
}
```

10.数字转换大小写问题

答案：

```
import java.util.Scanner;
public class Test1 {
    public static void main(String[] args){
        //键盘输入，转换成字符型数组
        Scanner sc = new Scanner(System.in);
        //为了提示，我们打印一句"请在键盘上输入一串数字："
        System.out.println("请在键盘上输入一串数字：");
        //接收字符串
        String str = sc.nextLine();
        //调用方法.toCharArray();将字符串转译成 char 型数组
        char[] num = str.toCharArray();
        //定义数组，将大写的数字也录入数组
        char[] China = {'零','一','二','三','四','五','六','七','八','九'};
        //遍历数组
```

```
                for (int i = 0; i < num.length; i++) {
                        for (int j = 0; j < China.length; j++) {
                        //用 num[i]-'0'得到 China[]脚标并打印相对应的元素
                                if (num[i]-'0'==j) {
                                //根据题目提示，输出时我们不需要换行
                                        System.out.print(China[j]);
                                }
                        }
                }
        }
}
```

11.数组去除重复问题

答案：

```
public static void moveSame(int[] arr){
                int[] newarr = null;
                ArrayList list = new ArrayList();
                for(int i : arr){
                        if(!list.contains(i)){
                                list.add(i);
                        }
                }
        }
```

12.希尔排序

答案：

```
public static void xierpaixu(int array[], int length) {// 10
                int len = length / 2;
                int i, j, temp;
                while (len >= 1) {
                // 每次都取它的中间下标， 循环的次数等于总长/2，除的次数
                        for (i = len; i < length; i++) {// 从 5 的下标循环到 9 下标
                                temp = array[i];// 保存值
                                j = i - len;// j 是从 0 的下标变换到下标 4
                                // 0 到 4 的下标，每个值和 temp 比较，交换 j 和 temp 的值
                                while (j >= 0 && array[j] > temp) {
                                // 如果 0 下标的值大于中间下标的值
                                        array[j + len] = array[j];// 0 下标的值给中间的角标
                                        j = j - len;// 改变 j 的下标，这时 j 为负数，跳出循环
                                }
                                // 把 j 变成 0 的下标 ,给下标 j 赋值 ,j 是当前循环时的下标，
                                array[j + len] = temp;
```

```
                }
                len = len / 2; // 缩小增量
        }
        for (int a = 0; a < length; a++) {// 遍历数组
                System.err.print(array[a] + ",");
        }
}
```