**尚学堂Java 初级软件工程师认证考试试卷**

**答案精讲**

1. **填空题（共20个题目，总计20分）**
2. public
3. switch
4. 数字
5. continue
6. static
7. 构造方法
8. 98
9. java.lang
10. Calendar
11. abstract
12. 声明数组
13. 异常处理
14. read
15. mkdir()
16. 阻塞
17. HashMap
18. null
19. 队列先进先出，堆栈先进后出
20. URL
21. 树
22. **选择题（共25个题目，总计25分）**
    1. C

分析：方法fun()使用了修饰符static，叫类方法，类方法是不能使用非static的属性，所以在属性前加static，修改为类变量。

* 1. A

分析：B应改为char c=s.charAt(1),答案C应改为int len=s.length，是使用了String类中的求字符串长度的方法，数组应该为.length属性，而字符串是length()，答案D，不可以将int类型的变量赋给String类型，应改为String t=”100”

* 1. B

分析:引用数据类型在作参数传递的时候传递的是对象的引用，指向同一块内存空间，当对象中的内容改变时，在方法结束后看到的是改变之后的结果

|  |
| --- |
|  |

* 1. C

分析：无论是否产生异常都执行的是finally。

* 1. D
  2. D

分析：

|  |
| --- |
|  |

* 1. BD

分析：方法重载的概念：

同：在同一个类中，方法名称相同

不同：参数的类型，个数，顺序不同

无关：与访问修饰符和返回值类型无关

* 1. C

分析：先将i的值赋给j，j的值为1,然后i再自加，i的值为2

* 1. A

分析：y<x++ ;结果是4<3 结果为false ,x++结果为4, 再将0赋值给x,结果为0

* 1. AD

分析：一个try块后可以有多个catch块，称为多重catch。try,catch,finally的组合形式为try-catch,try-finally,try-catch-finally。所以finally是不可以单独使用的。

* 1. D

分析：使用this调用本类带有两个参数的构造方法。

* 1. D
  2. AC

分析：含有抽象方法的类，必须是抽象类，反过来，抽象类中可以有0个或多个抽象方法，也可以有普通方法。子类必须重写父类的所有抽象方法，除非子类也是抽象类。

* 1. B

分析：

|  |
| --- |
|  |

* 1. B

分析：数组的长度是length属性，数组的下标从0开始，最大下标为 数组名.length-1

* 1. AB

分析：

|  |
| --- |
|  |

* 1. BC

分析：//2盘符与文件之间的分隔符应改为c:\\mytext.txt，转义字符。

//3方法写错，没有这个方法，应改为fw.write(str,0,str.length()); fw.flush;

* 1. B

分析：File类的方法

* 1. B

分析: ClassNotFoundException:无法找到指定的类异常，应用程序试图使用如Class类中的forName方法通过字符串名称加载类时，抛出该异常。

IOException：当发生某种 I/O 异常时，抛出此异常。此类是失败或中断的 I/O 操作生成的异常的通用类。

SQLException：提供关于数据库访问错误或其他错误信息的异常。

RemoteException：是许多与通信相关的异常的通用超类，这些异常可能会在执行远程方法调用期间发生。

* 1. C

分析：Map集合是键值对的形式存储数据，要求存储的键不能相同，如果存储了同相的键，那么后添加进入的会覆盖先添加的数据，所以集合的长度为1，集合中元素的值为 2。

* 1. AC

分析：异常关键字的用法，throws用于抛出方法的异常，并且由该方法调用处的异常处理类来处理该异常 ，它常用在方法的后面，throw用于方法块里面的代码，比throws的层次要低，表示它抛出异常，但它不会处理它，而是由方法块的throws Exception来调用异常处理类来处理。

* 1. AD
  2. A

分析:

|  |
| --- |
|  |

* 1. D

分析：

|  |
| --- |
|  |

* 1. C

分析：join方法:Java中当前运行的线程可以调用另外一个线程的join()方法，使当前的线程转到阻塞状态，等待另一个线程运行完毕再继续运行。

interrupt()只是改变中断状态而已 ，interrupt()不会中断一个正在运行的线程。这一方法实际上完成的是，在线程受到阻塞时抛出一个中断信号，这样线程就得以退出阻塞的状态。更确切的说，如果线程被Object.wait, Thread.join和Thread.sleep三种方法之一阻塞，那么，它将接收到一个中断异常（InterruptedException），从而提早地终结被阻塞状态。 如果线程没有被阻塞，这时调用interrupt()将不起作用。

isAlive方法：当调用start()方法后，线程开始执行方法中的代码。线程处于运行状态，Thread类的isAlive方法来判断线程是否处于运行状态。

currentThread方法：返回当前线程的引用。

1. **判断题（共20个题目，总计10分）**
2. ×

分析：方法Integer.parseInt将一个String转变成整数。

1. √
2. ×

分析: 因为数组的单个基本数据类型以值调用方法传递。如果将整个数组传递给方法， 数组元素的任何变量都将反映在原来数组上。另外，如果数组单个元素是类的对象，传递单个元素也引用调用形式传递给方法，对象的改变将反映在原数组元素上。

1. √
2. ×

分析：t.length等于2获得二维数组中一维的长度，t[0].length等于3获得是二维数组中第二维中第一数组的长度

1. √
2. ×

分析:布尔变量的值为false

1. ×

分析：类的实例变量一般设为private

1. ×

分析：线程只能用yield使用相同优先级的程序运行。

1. ×

分析：如果这个类没有使用extends关键字直接继承父类，那么这个类默认继承自Object类

1. ×

分析:它可以处理在java源程序中介于”/\*”和”\*/”之间的注释，并生成相应的程序文档。

1. ×

分析：out是System类中的一个静态常量，定义为:public static final PrintStream out

1. √
2. ×

分析：Vector只能存储Object。必须使用java.lang包中的类型包装类创建。

1. ×

分析：子类不能继承超类的构造函数，然而子类构造函数可以通过super引用调用超类的构造函数。

1. ×

分析：Java是对大小写字母敏感的，所以这些变量不相同。

1. ×

分析：在块中声明的变量或是在方法的参数表中声明的变量都具有自动生命期。

1. √
2. √
3. √
4. **简答题（共5个题目，总计25分）**

**1.final、finally、finalize的区别。（3分）**

* + - * final用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。
      * finally是异常处理语句结构的一部分，表示总是执行。
      * finalize是Object类的一个方法，在垃圾收集器执行的时候会调用回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。

**评分要求：共计3项，每项1分。**

**2.Error和Exception的区别， Checked异常和Runtime异常的区别。（6分）**

**Error和Exception的区别**

* Error类，表示仅靠程序本身无法恢复的严重错误，比如说内存溢出、动态链接异常、虚拟机错误。应用程序不应该抛出这种类型的对象。假如出现这种错误，除了尽力使程序安全退出外，在其他方面是无能为力的。所以在进行程序设计时，应该更关注Exception类。
* Exception类，由Java应用程序抛出和处理的非严重错误，比如所需文件没有找到、零作除数，数组下标越界等。它的各种不同子类分别对应不同类型异常。可分为两类：Checked异常和Runtime异常。

**Checked异常和Runtime异常的区别**

* 运行时异常：包括RuntimeaException及其所有子类。不要求程序必须对它们作出处理，比如InputMismatchException、ArithmeticException、NullPointerException等。即使没有使用try-catch或throws进行处理，仍旧可以进行编译和运行。如果运行时发生异常，会输出异常的堆栈信息并中止程序执行。
* Checked异常（非运行时异常）：除了运行时异常外的其他异常类都是Checked异常。程序必须捕获或者声明抛出这种异常，否则出现编译错误，无法通过编译。处理方式包括两种：通过try-catch捕获异常，通过throws声明抛出异常从而交给上一级调用方法处理。

**评分要求：Error和Exception的区别，每项1.5分共3分**

**Checked异常和Runtime异常的区别，每项1.5分共3分。**

**3.String、StringBuffer、StringBuilder区别与联系。（5分）**

* String类是不可变类，即一旦一个String对象被创建后，包含在这个对象中的字符序列是不可改变的，直至这个对象销毁。
* StringBuffer类则代表一个字符序列可变的字符串，可以通过append、insert、reverse、setChartAt、setLength等方法改变其内容。一旦生成了最终的字符串，调用toString方法将其转变为String
* JDK1.5新增了一个StringBuilder类，与StringBuffer相似，构造方法和方法基本相同。不同是StringBuffer是线程安全的，而StringBuilder是线程不安全的，所以性能略高。通常情况下，创建一个内容可变的字符串，应该优先考虑使用StringBuilder

**评分要求：前两项每项1.5分，第三项2分，本题共5分。**

**4. HashSet采用了哈希表作为存储结构，请说明哈希表的特点和实现原理。**

**提示：结合Object类的hashCode()和equals()说明其原理（6分）**

* 哈希表的查询速度特别快，时间复杂度为O。
* HashMap、Hashtable、HashSet这些集合采用的是哈希表结构，需要用到hashCode哈希码，hashCode是一个整数值。
* 系统类已经覆盖了hashCode方法 自定义类如果要放入hash类集合，必须重写hashcode。如果不重写，调用的是Object的hashcode，而Object的hashCode实际上是地址。
* 向哈希表中添加数据的原理：当向集合Set中增加对象时，首先集合计算要增加对象的hashCode码，根据该值来得到一个位置用来存放当前对象，如在该位置没有一个对象存在的话，那么集合Set认为该对象在集合中不存在，直接增加进去。如果在该位置有一个对象存在的话，接着将准备增加到集合中的对象与该位置上的对象进行equals方法比较，如果该equals方法返回false,那么集合认为集合中不存在该对象，在进行一次散列，将该对象放到散列后计算出的新地址里。如果equals方法返回true，那么集合认为集合中已经存在该对象了，不会再将该对象增加到集合中了。
* 在哈希表中判断两个元素是否重复要使用到hashCode()和equals()。hashCode决定数据在表中的存储位置，而equals判断是否存在相同数据。
* Y=K(X) ：K是函数，X是哈希码，Y是地址

**评分要求：前三项每项1分，共3分，原理2分，后两项，每项各0.5分，**

**本题共6分。**

**5. 实现Java反射技术的主要类有哪些，其作用分别是什么？（5分）**

在JDK中，主要由以下类来实现Java反射机制，这些类都位于java.lang.reflect包中

* Class类：代表一个类
* Field 类：代表类的成员变量(属性)
* Method类：代表类的成员方法
* Constructor 类：代表类的构造方法
* Array类：提供了动态创建数组，以及访问数组的元素的静态方法

**评分要求：每项1分，类与作用各占0.5分，本题共5分。**

1. **编码题（共3个题目，总计20分）**

1. **编写应用程序，创建类的对象，分别设置圆的半径、圆柱体的高，计算并分别显示圆半径、圆面积、圆周长，圆柱体的体积。（7分）**

（1）编写一个圆类Circle，该类拥有：

* + - 1. 一个成员变量

Radius（私有，浮点型）;//存放圆的半径；

* + - 1. 两个构造方法

Circle（） //将半径设为0

Circle（double r ） //创建Circle对象时将半径初始化为r

* + - 1. 三个成员方法

double getArea（） //获取圆的面积

double getPerimeter（） //获取圆的周长

void show（） //将圆的关径、周长、面积输出到屏幕

（2） 编写一个圆柱体类Cylinder，它继承于上面的Circle类。还拥有：

* + 1. 一个成员变量

double hight （私有，浮点型）; //圆柱体的高；

* + 1. 构造方法

Cylinder（double r,double h） //创建Cylinder对象时将半径初始化为r,高度初始化为h

* + 1. 成员方法

double getVolume（） //获取圆柱体的体积

void showVolume（） //将圆柱体的体积输出到屏幕

**答案：**

1. **圆类**

|  |
| --- |
| **public** **class** Circle { //定义父类--圆类  //私有属性-🡪0.5分  **private** **double** radius; //成员变量--圆半径  //两个构造方法，共0.5分，只对一项不得分  Circle(){ //构造方法  radius=0.0;  }  Circle(**double** r){ //构造方法  radius=r;  }  // 方法定义及返回值正确🡪0.5分，只对一项不得分  **double** getPerimeter(){ //成员方法--求圆周长  **return** 2\*Math.*PI*\*radius;  }  // 方法定义及返回值正确🡪0.5分，只对一项不得分  **double** getArea(){ //成员方法--求圆面积  **return** Math.*PI*\*radius\*radius;  }  // 方法定义正确及方法调用正确🡪0.5分，只对一项不得分  **void** show(){ //成员方法--显示圆半径、周长、面积  System.*out*.println("圆半径="+radius);  System.*out*.println("圆周长="+getPerimeter());  System.*out*.println("圆面积="+getArea());  }  } |

1. **圆柱类**

|  |
| --- |
| //正确继承父类🡪0.5分  **public** **class** Cylinder **extends** Circle { //定义子类--圆柱类  //私有属性 🡪0.5分  **private** **double** hight; //定义成员变量--圆柱高  //构造方法定义及调用父类构造方法正确🡪0.5 只对一项不得分  Cylinder(**double** r,**double** h) { //构造方法  **super**(r);  hight=h;  }  //方法定义及返回值正确🡪0.5分，只对一项不得分  **public** **double** getVolume(){ //成员方法--求圆柱体积  **return** getArea()\*hight;  }  // 方法定义正确及方法调用正确🡪0.5分，只对一项不得分  **public** **void** showVolume(){ //成员方法--显示圆柱体积  System.*out*.println("圆柱体积="+getVolume());  }  } |

1. **测试主类**

|  |
| --- |
| **public** **class** TestCylinder { //定义主类  **public** **static** **void** main(String[] args) { //主程序入口  //创建圆类对象正确及调用方法正确🡪1分  Circle ci=**new** Circle(10.0); //生成圆类实例  ci.show(); //调用圆类的方法  //创建圆类对象正确及调用方法正确🡪1分  Cylinder cyl=**new** Cylinder(5.0, 10.0); //生成圆柱类实例  cyl.show(); //调用父类的方法  cyl.showVolume(); //调用子类方法  }  } |

1. **设计4个线程，其中两个线程每次对j增加1，另外两个线程对j每次减少1。**

**编写实现，要求：使用内部类实现线程，对j增减的时候不考虑顺序问题。（6分）**

|  |
| --- |
| **public** **class** ThreadTest { //创建类  **private** **int** j; //声明成员变量j  **public** **static** **void** main(String[] args) { //程序主入口  ThreadTest tt=**new** ThreadTest(); //创建ThreadTest的对象  **//创建内部类对象正确各0.5分，共1分**  Inc inc=tt.**new** Inc(); //创建内部线程类对象  Dec dec=tt.**new** Dec(); //创建内部线程类对象  **//如不使用循环，而是手动创建4个线程，可适当给分**  **for**(**int** i=0;i<2;i++){  **// 创建线程对象正确，启动线程各0.5分共1分**  Thread t=**new** Thread(inc);  t.start();  **// 创建线程对象正确，启动线程各0.5分共1分**  t=**new** Thread(dec);  t.start();  }  }  **//实现同步对j的值加，1分**  **private** **synchronized** **void** inc(){  j++; //j++  System.*out*.println(Thread.*currentThread*().getName()+"-inc:"+j);  }  **//实现同步对j的值减，1分**  **private** **synchronized** **void** dec(){  j--;  System.*out*.println(Thread.*currentThread*().getName()+"-dec:"+j);  }  //内部类  **//内部类定义正确并实现run方法1分**  **class** Inc **implements** Runnable{  @Override  **public** **void** run() {  **for** (**int** i = 0; i <100; i++) {  inc(); //调用加的方法  }  }  }  **//内部类定义正确并实现run 方法1分**  **class** Dec **implements** Runnable{  @Override  **public** **void** run() {  **for** (**int** i = 0; i <100; i++) {  dec(); //调用减的方法  }  }  }  } |

1. **Java的通信编程，用Java Socket编程，要求从客户端录入几个字符，发送到服务器端，由服务器端将接收到的字符进行输出。（7分）**

**提示:**

**服务器端：**PrintWriter out =**new** PrintWriter(socket.getOutputStream(),**true**);

**客户端:**

BufferedReader line=**new** BufferedReader(**new** InputStreamReader(System.*in*));

**答:**

**服务器端：（4分）**

|  |
| --- |
| **import** java.io.\*;  **import** java.net.\*;  **public** **class** Server { //服务器端类  **private** ServerSocket ss; //声明ServerSocket对象  **private** Socket socket; //声明Socket对象  **private** BufferedReader in; //声明输入流对象BufferedReader  **private** PrintWriter out; //声明输出流对象  **public** Server(){  //异常处理0.25分  **try** {  //创建一个ServerSocket，指定等待端口🡪1分  ss=**new** ServerSocket(10000);  **while**(**true**){  //使用ServerSocket接收用户请求（处于监听状态）🡪1分  socket=ss.accept(); //获得客户端的socket对象  //创建输入流并读取信息🡪0.5分  in=**new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  String line=in.readLine();  System.*out*.println("客户端说:"+line);  //创建输出流并输出信息🡪0.5分  out=**new** PrintWriter(socket.getOutputStream(),**true**);  out.print("您的信息已接收!");  //关闭流和socket🡪0.5分  out.close();  in.close();  socket.close();  }  } **catch** (IOException e) {  out.println("出错了!");  } //创建ServerSocket对象  }  **public** **static** **void** main(String[] args) {//程序主入口  //创建对象🡪0.25分  **new** Server();  }  } |

**客户端：（3分）**

|  |
| --- |
| **import** java.io.\*;  **public** **class** Client { //声明客户端类  Socket socket; //声明Socket对象  BufferedReader in; //声明输入流对象  PrintWriter out; //声明输出流对象  **public** Client(){ //构造方法  //异常处理0.25分  **try** {  //创建Socket对象，用来发送请求🡪0.5分  socket=**new** Socket("127.0.0.1", 10000);  //创建字符缓冲区对象🡪0.5分  BufferedReader line  =**new** BufferedReader(**new** InputStreamReader(System.*in*));  //创建输出流对象并输出🡪0.5分  out=**new** PrintWriter(socket.getOutputStream(),**true**);  out.println(line.readLine());  //创建输入流对象🡪0.5分  in=**new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  System.*out*.println(in.readLine());  //关闭流对象🡪0.5分  out.close();  in.close();  } **catch** (Exception e) {  System.*out*.println("出错了!");  }  }  **public** **static** **void** main(String[] args) {  //创建对象🡪0.25分  **new** Client();  }  } |