**测试**

1. **填空题**
2. 在Java中每个Java基本类型在java.lang包中都在一个相应的包装类，把基本类型数据转换为对象，其中包装类Integer是\_\_\_\_\_\_\_\_\_\_\_\_的直接子类。
3. 包装类Integer的静态方法可以将字符串类型的数字”123”转换成基本整型变量n，其实现语句是：\_\_\_\_\_\_\_\_\_\_\_\_。
4. 在Java中使用java.lang包中的\_\_\_\_\_\_\_\_\_\_\_\_类来创建一个字符串对象，它代表一个字符序列可变的字符串，可以通过相应的方法改变这个字符串对象的字符序列。
5. StringBuilder类是StringBuffer类的替代类，两者的共同点是都是可变长度字符串，其中线程安全的类是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
6. DateFormat类可以实现字符串和日期类型之间的格式转换，其中将日期类型转换为指定的字符串格式的方法名是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
7. 使用Math.random( )返回带正号的 double值，该值大于等于0.0且小于1.0。使用该函数生成[30,60]之间的随机整数的语句是\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
8. JDK1.5后提供了\_\_\_\_\_\_\_\_\_\_\_\_关键字，用以定义枚举类。枚举类是一种特殊的类，可以有自己的属性、方法和构造方法。
9. File对象调用方法 创建一个目录，不包括所有必需但不存在的父目录，当且仅当已创建目录时，返回true；否则返回false。
10. \_\_\_\_\_\_\_\_\_\_\_\_机制是一种非常有用的辅助性程序设计方法。采用这种方法可以使得在程序设计时将程序的正常流程与错误处理分开，有利于代码的编写和维护。
11. 在Java异常处理中可以使用多个catch子句，此时包含异常类的父类Exception的catch子句的位置应该是在\_\_\_\_\_\_\_\_\_\_\_\_。
12. 异常处理中finally块可确保无论是否发生异常，该块中代码总能被执行。finally块不执行的唯一情况是在异常处理代码中执行\_\_\_\_\_\_\_\_\_\_\_\_语句退出Java虚拟机。
13. 异常是由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，可分为两类：Checked异常和\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
14. 在Java中对于程序可能出现的检查时异常，要么用try…catch语句捕获并处理它，要么使用\_\_\_\_\_\_\_\_\_\_\_\_语句抛出它，由上一级调用者来处理。
15. Java异常处理中，如果一个方法中出现了多个Checked异常，可以在方法声明中使用关键字\_\_\_\_\_\_\_\_\_\_\_\_\_\_声明抛出，各异常类型之间使用逗号分隔。
16. **选择题**

|  |  |  |
| --- | --- | --- |
| **1.** | **Person类和Test类的代码如下所示，则代码中的错误语句是（ ）。（选择一项）** | |
|  | **public** **class** Person {  public String name;  **public** Person(String name) {  **this**.name = name;  }  }  **public** **class** Test {  **public** **static** **void** main(String[] args) {  **final** Person person = **new** Person("欧欧");  person.name = "美美";  person = **new** Person("亚亚");  }  } | |
|  |  |  |
|  | **A.** | final Person person = new Person("欧欧"); |
|  | **B.** | person.name = "美美"; |
|  | **C.** | person = new Person("亚亚"); |
|  | **D.** | 没有错误 |

|  |  |  |
| --- | --- | --- |
| **2.** | **如下Java代码运行的结果是（ ）。（选择一项）** | |
|  | **public** **class** Test {  **final** **int** age;  **public** Test(){  age=10;  }  **public** **static** **void** main(String[ ] args) {  System.*out*.println(**new** Test().age);  }  } | |
|  |  |  |
|  | **A** | 输出：0 |
|  | **B.** | 输出：10 |
|  | **C.** | 输出：null |
|  | **D.** | 运行时出现异常 |

|  |  |  |
| --- | --- | --- |
| **3.** | **以下代码中错误的语句是（ ）。（选择一项）** | |
|  | **public** **class** Something{  **public** **static** **void** main(String[] args){  **final** Other o=**new** Other();  **new** Something().addOne(o);//1  }  **public** **void** addOne( Other o){  o.i++;//2  o = **new** Other();//3  }  }  **class** Other{  **public** **int** i;  } | |
|  |  |  |
|  | **A** | 1 |
|  | **B.** | 2 |
|  | **C.** | 3 |
|  | **D.** | 没有错误 |

|  |  |  |
| --- | --- | --- |
| **4.** | **下列选项中，关于Java的抽象类和抽象方法说法正确的是（ ）。（选择二项）** | |
|  |  |  |
|  | **A** | 抽象类中可以含有0个或多个抽象方法 |
|  | **B.** | 抽象类中不可以有构造方法 |
|  | **C.** | 一个类中若有抽象方法，则这个类必为抽象类 |
|  | **D.** | 子类必须重写父类所有的抽象方法 |

|  |  |  |
| --- | --- | --- |
| **5.** | **在Java中关于abstract关键字，以下说法正确的是（ ）。（选择两项）** | |
|  |  |  |
|  | **A** | abstract类中可以没有抽象方法 |
|  | **B.** | abstract类的子类也可以是抽象类 |
|  | **C.** | abstract方法可以有方法体 |
|  | **D.** | abstract类可以创建对象 |

|  |  |  |
| --- | --- | --- |
| **6.** | **以下选项可替换题目中//add code here而不产生编译错误的是（ ）。（选择二项）** | |
|  | **public** **abstract** **class** MyClass{  **public** **int** constInt=5;  //add code here  **public** **void** method( ){  }  } | |
|  |  |  |
|  | **A** | **public** **abstract** **void** method(**int** a); |
|  | **B.** | constInt =constInt+5; |
|  | **C.** | **public** **int** method( ); |
|  | **D.** | **public** **abstract** **void** anotherMethod( ); |

|  |  |  |
| --- | --- | --- |
| **7.** | **在Java接口中，下列选项中属于有效的方法声明是（ ）。（选择二项）** | |
|  |  |  |
|  | **A** | public void aMethod( ); |
|  | **B.** | final void aMethod( ); |
|  | **C.** | void aMethod(){ } |
|  | **D.** | private void aMethod( ); |

|  |  |  |
| --- | --- | --- |
| **8.** | **以下选项中关于匿名内部类的说法正确的是（ ）。（选择二项）** | |
|  |  |  |
|  | **A.** | 匿名内部类可以实现多个接口，或者继承一个父类 |
|  | **B.** | 匿名内部类不能是抽象类，必须实现它的抽象父类或者接口里包含的所有抽象方法 |
|  | **C.** | 匿名内部类没有类名，所以匿名内部类不能定义构造方法 |
|  | **D.** | 匿名内部类可以直接访问外部类的所有局部变量 |

1. **选择题2**

|  |  |  |
| --- | --- | --- |
| **1.** | **以下选项中关于int和Integer的说法错误的是（ ）。（选择二项）** | |
|  |  |  |
|  | **A.** | int是基本数据类型，Integer是int的包装类，是引用数据类型 |
|  | **B.** | int的默认值是0，Integer的默认值也是0 |
|  | **C.** | Integer可以封装了属性和方法提供更多的功能 |
|  | **D.** | Integer i=5;该语句在JDK1.5之后可以正确执行，使用了自动拆箱功能 |

|  |  |  |
| --- | --- | --- |
| **2.** | **分析如下Java代码，该程序编译后的运行结果是（ ）。（选择一项）** | |
|  | **public** **static** **void** main(String[ ] args) {  String str=**null**;  str.concat("abc");  str.concat("def");  System.*out*.println(str);  } | |
|  |  |  |
|  | **A** | null |
|  | **B.** | abcdef |
|  | **C.** | 编译错误 |
|  | **D.** | 运行时出现NullPointerException异常 |

|  |  |  |
| --- | --- | --- |
| **3.** | **以下关于String类的代码的执行结果是（ ）。（选择一项）** | |
|  | **public** **class** Test2 {  **public** **static** **void** main(String args[]) {  String s1 = **new** String("hello");  String s2 = **new** String("hello");  **if** (s1 == s2)  System.*out*.println("s1 == s2");  **if** (s1.equals(s2))  System.*out*.println("s1.equals(s2)");  }  } | |
|  |  |  |
|  | **A.** | s1 == s2 |
|  | **B.** | s1.equals(s2) |
|  | **C.** | s1 == s2  s1.equals(s2) |
|  | **D.** | 以上都不对 |

|  |  |  |
| --- | --- | --- |
| **4.** | **以下关于StringBuffer类的代码的执行结果是（ ）。（选择一项）** | |
|  | **public** **class** **TestStringBuffer {**  **public** **static** **void** main(String args[]) {  StringBuffer a = **new** StringBuffer("A");  StringBuffer b = **new** StringBuffer("B");  *mb\_operate*(a, b);  System.*out*.println(a + "." + b);  }  **static** **void** mb\_operate(StringBuffer x, StringBuffer y) {  x.append(y);  y = x;  }  } | |
|  |  |  |
|  | **A.** | A.B |
|  | **B.** | A.A |
|  | **C.** | AB.AB |
|  | **D.** | AB.B |

|  |  |  |
| --- | --- | --- |
| **5.** | **给定如下Java代码，编译运行的结果是（ ）。（选择一项）** | |
|  | **public** **static** **void** main(String []args){  String s1= **new** String("pb\_java\_OOP\_T5");  String s2 = s1.substring(s1.lastIndexOf("\_"));  System.*out*.println("s2="+s2);  } | |
|  |  |  |
|  | **A** | s2=\_java\_OOP\_T5 |
|  | **B.** | s2=\_OOP\_T5 |
|  | **C.** | s2=\_T5 |
|  | **D.** | 编译出错 |

|  |  |  |
| --- | --- | --- |
| **6.** | **对于语句String s="my name is kitty"，以下选项中可以从其中截取”kitty”的是（ ）（选择二项）** | |
|  |  |  |
|  | **A** | s.substring(11,16) |
|  | **B.** | s.substring(11) |
|  | **C.** | s.substring(12,17) |
|  | **D.** | s.substring(12,16) |

|  |  |  |
| --- | --- | --- |
| **7.** | **分析下面的Java程序段，编译运行后的输出结果是（ ）。（选择一项）** | |
|  | **public** **class** Test {  **public** **void** changeString(StringBuffer sb) {  sb.append("stringbuffer2");  }  **public** **static** **void** main(String[] args) {  Test a = **new** Test();  StringBuffer sb = **new** StringBuffer("stringbuffer1");  a.changeString(sb);  System.*out*.println("sb = " + sb);  }  } | |
|  |  |  |
|  | **A** | sb = stringbuffer2stringbuffer1 |
|  | **B.** | sb = stringbuffer1 |
|  | **C.** | sb = stringbuffer2 |
|  | **D.** | sb = stringbuffer1stringbuffer2 |

|  |  |  |
| --- | --- | --- |
| **8.** | **给定如下Java代码，编译运行的结果是（ ）。（选择一项）** | |
|  | **public** **static** **void** main(String[] args) {  StringBuffer sbf = **new** StringBuffer("java");  StringBuffer sbf1 = sbf.append(",C#");  String sbf2 = sbf + ",C#";  System.*out*.print(sbf.equals(sbf1));  System.*out*.println(sbf2.equals(sbf));  } | |
|  |  |  |
|  | **A** | true false |
|  | **B.** | true true |
|  | **C.** | false false |
|  | **D.** | false true |

|  |  |  |
| --- | --- | --- |
| **9.** | **分析下面的Java程序，编译运行后的输出结果是（ ）。（选择一项）** | |
|  | **public** **class** Example {  String str = **new** String("good");  **char**[] ch = { 'a', 'b', 'c' };  **public** **static** **void** main(String args[]) {  Example ex = **new** Example( );  ex.change(ex.str, ex.ch);  System.*out*.print(ex.str + "and");  System.*out*.print(ex.ch);  }  **public** **void** change(String str, **char** ch[]) {  str = "test ok";  ch[0] = 'g';  }  } | |
|  |  |  |
|  | **A** | goodandabc |
|  | **B.** | goodandgbc |
|  | **C.** | test okandabc |
|  | **D.** | test okandgbc |

|  |  |  |
| --- | --- | --- |
| **10.** | **以下程序片段中可以正常编译的是（ ）。（选择一项）** | |
|  |  |  |
|  | **A** | String s = "Gone with the wind";  String k = s+t;  String t = "good"; |
|  | **B.** | String s = "Gone with the wind";  String t;  t = s[3]+"one"; |
|  | **C.** | String s = "Gone with the wind";  String stanfard = s.toUpperCase(); |
|  | **D.** | String s = "home directory";  String t = s – "directory"; |

|  |  |  |
| --- | --- | --- |
| **11.** | **File类中的（ ）方法可以用来判断文件或目录是否存在。（选择一项）** | |
|  |  |  |
|  | **A** | exist() |
|  | **B.** | exists() |
|  | **C.** | fileExist() |
|  | **D.** | fileExists() |

|  |  |  |
| --- | --- | --- |
| **12.** | **在Java中，以下File类的方法中（ ）用来判断是否是目录。（选择一项）** | |
|  |  |  |
|  | **A** | isFile( ) |
|  | **B.** | getFile( ) |
|  | **C.** | isDirectory( ) |
|  | **D.** | getPath( ) |
|  |  |  |

1. **选择题3**

|  |  |  |
| --- | --- | --- |
| **1.** | **以下关于异常的代码的执行结果是（ ）。（选择一项）** | |
|  | **public** **class** Test {  **public** **static** **void** main(String args[]) {  **try** {  System.*out*.println("try");  **return**;  } **catch**(Exception e){  System.*out*.println("catch");  }**finally** {  System.*out*.println("finally");  }  }  } | |
|  |  |  |
|  | **A.** | try  catch  finally |
|  | **B.** | catch  finally |
|  | **C.** | try  finally |
|  | **D.** | try |

|  |  |  |
| --- | --- | --- |
| **2.** | **在异常处理中，如释放资源、关闭文件等由（ ）来完成。（选择一项）** | |
|  |  |  |
|  | **A** | try子句 |
|  | **B.** | catch子句 |
|  | **C.** | finally子句 |
|  | **D.** | throw子句 |

|  |  |  |
| --- | --- | --- |
| **3.** | **编译并运行如下Java程序，将输出（ ）。（选择一项）** | |
|  | **public** **static** **void** main(String[] args) {  **try** {  **int** num1 = 2; **int** num2 = 0;  **int** result = num1 / num2;  System.*out*.println(result);  **throw** **new** NumberFormatException( );  } **catch** (ArrayIndexOutOfBoundsException e) {  System.*out*.print("1");  } **catch** (NumberFormatException e) {  System.*out*.print("2");  } **catch** (Exception e) {  System.*out*.print("3");  } **finally** {  System.*out*.print("4");  }  System.*out*.print("5");  } | |
|  |  |  |
|  | **A** | 134 |
|  | **B.** | 2345 |
|  | **C.** | 1345 |
|  | **D.** | 345 |

|  |  |  |
| --- | --- | --- |
| **4.** | **阅读如下Java代码,其中错误的行是（ ）。（选择二项）** | |
|  | **public** **class** Student {  **private** String stuId;  **public** **void** setStuId(String stuId) **throw** Exception { // 1  **if** (stuId.length() != 4) { // 2  **throws** **new** Exception("学号必须为4位!"); // 3  } **else** {  **this**.stuId = stuId; //4  }  }  } | |
|  |  |  |
|  | **A** | 1 |
|  | **B.** | 2 |
|  | **C.** | 3 |
|  | **D.** | 全部正确 |

|  |  |  |
| --- | --- | --- |
| **5.** | **下面选项中有关Java异常处理模型的说法错误的是（ ）。（选择二项）** | |
|  |  |  |
|  | **A** | 一个try块只能有一条catch语句 |
|  | **B.** | 一个try块中可以不使用catch语句 |
|  | **C.** | catch块不能单独使用，必须始终与try块在一起 |
|  | **D.** | finally块可以单独使用，不是必须与try块在一起 |

|  |  |  |
| --- | --- | --- |
| **6.** | **下面选项中属于运行时异常的是（ ）。（选择二项）** | |
|  |  |  |
|  | **A** | Exception和SexException |
|  | **B.** | NullPointerException和InputMismatchException |
|  | **C.** | ArithmeticException和ArrayIndexOutOfBoundsException |
|  | **D.** | ClassNotFoundException和ClassCastException |

|  |  |  |
| --- | --- | --- |
| **7.** | **阅读如下Java代码，在控制台输入"-1",执行结果是（）。（选择一项）** | |
|  | **public** **class** Demo {  **public** **static** **void** main(String[] args) {  Scanner input = **new** Scanner(System.***in***);  System.***out***.print("请输入数字：");  **try** {  **int** num = input.nextInt();  **if** (num < 1 || num > 4) {  **throw** **new** Exception("必须在1-4之间！");  }  } **catch** (InputMismatchException e) {  System.***out***.println("InputMismatchException");  } **catch** (Exception e) {  System.***out***.println(e.getMessage());  }  }  } | |
|  |  |  |
|  | **A** | 输出：InputMismatchException |
|  | **B.** | 输出：必须在1-4之间！ |
|  | **C.** | 什么也没输出 |
|  | **D.** | 编译错误 |

1. **判断题**
2. 声明为final的类不能是超类。（ ）
3. 使用final修饰的变量将变成常量，其中不能再被改变；使用final修饰的方法将无法被子类重载；使用final修饰的类将无法再被其他类继承。（ ）
4. 抽象方法不能含有方法体，并且必须定义在抽象类中。（ ）
5. 抽象类是指在class前加使用abstract关键字修饰，且可以存在抽象方法和普通方法的类。（ ）
6. 接口中只有常量，没有变量；只有抽象方法，并且全部都是public方法。（ ）
7. 抽象类和接口都不能实例化，都没有构造方法。（ ）
8. 接口并不要求实现类和接口在概念本质上一致的，仅仅是实现了接口定义的约定或者能力而已。接口定义了“做什么”，而实现类负责完成“怎么做”，体现了功能（规范）和实现分离的原则。（ ）
9. 内部类作为外部类成员，权限修饰符和其他成员一样，可声明为private、默认、protected或public。( )
10. 匿名内部类适合创建那种只需要使用一次的类，它可以实现一个或多个接口，或者继承一个父类。（ ）
11. 对于物理连接，比如数据库连接、输入流输出流、Socket连接等，垃圾回收机制无能为力，必须手动关闭才可以。( )
12. 垃圾回收机制回收任何对象之前，总会先调用它gc( )方法，该方法是Object类提供的方法。不要主动调用某个对象的该方法，应该交给垃圾回收机制调用。（ ）
13. **判断题2**
14. 方法Integer.parseInt()的作用是将一个整数转变成String。（ ）
15. JK1.5后提供了自动装箱和自动拆箱功能，从而可以实现基本数据类型和对应包装类之间的自动转换，简化了操作。（ ）
16. 执行语句String str="abcedf"; int len=str.length; 后，能够得到字符串的长度是6。（ ）
17. 运算符“==”用于比较引用时，如果两个引用指向内存同一个对象，则返回true。（ ）
18. java.sql.Date类和java.util.Date类的关系是前者是后者的父类，其中前者没有提供无参数构造方法，而后者可以提供无参数构造方法来获取当前时间。（ ）
19. 求x的y次方，其表达式为：Math.pow(x,y)。（ ）
20. 一个File对象可以代表一个文件或目录，它可以获取文件和目录属性，也可以访问文件内容。( )
21. 在使用File类中的delete( )方法时，删除时可能文件不存在，所以我们最好先判断一下是否存在，不然会出现NullPointerException异常。（ ）
22. **编程题**
23. 接收从键盘输入的字符串格式的年龄，分数和入学时间，转换为整数、浮点数、日期类型，并在控制台输出。

提示：使用包装类Integer、Double和日期转换类DateFormat实现

1. 编写一个计算N个整数平均值的程序。程序应该提示用户输入N的值，如何必须输入所有N个数。如果用户输入的值是一个负数，则应该抛出一个异常并捕获，提示“N必须是正数或者0”。并提示用户再次输入该数
2. 由控制台按照固定格式输入学生信息，包括学号，姓名，年龄信息，当输入的内容为exit退出；将输入的学生信息分别封装到一个Student对象中，再将每个Student对象加入到一个集合中，要求集合中的元素按照年龄大小正序排序；最后遍历集合，将集合中学生信息输出到控制台，每个学生数据占单独一行
3. 实现不同符合PCI规范的适配器

需求说明：PCI是一种规范，所有实现了该规范的适配器，必如显卡、声卡、网卡都可以安装到PCI插槽上并工作。模拟实现该功能。



实现思路及关键代码

1. 定义PCI接口，具有传送数据send()方法
2. 定义显卡VideaCard类，实现该接口
3. 定义声卡AudioCard类，实现PCI接口
4. 定义网卡NetCard类，实现PCI接口
5. 定义测试类，让显卡、声卡、网卡发送数据

5.打印某个月份的可视化日历

1. 