



目录

下篇 知识点学习.....	2
第3章 对象声明与初始化.....	2
第4章 运算.....	9
第5章 垃圾收集.....	17
第6章 流程控制和差错处理.....	22
第7章 访问控制、继承和多态.....	29
第8章 并发与多线程.....	35
第9章 UTIL库.....	39
第10章 JAVA开发环境.....	42
第11章 JAVA的GUI库.....	45
第12章 WEB.....	50

下篇 知识点学习

第3章 对象声明与初始化

编号	试 题	知识点（考察点）说明
1	<pre>abstract class T1 { T1(){} abstract void method(){}; static abstract void method2(); }</pre> <p>以上代码中的三个方法，说法正确的是：（）</p> <p>A. 只有 T1()方法存在错误；</p> <p>B. 只有 method（）存在错误；</p> <p>C. 只有 method2()存在错误；</p> <p>D. 全部正确；</p> <p>E. 全部错误；</p> <p>F. 只有 method()和 method2()存在错误</p>	方法的声明

2	<pre> public class TT { private String instanceVar = "1"; private static String staticVar = "A"; static { instanceVar += "3"; staticVar += "B"; } TT() { instanceVar += "2"; staticVar += "C"; } public static void main(String args[]) { System.out.print(instanceVar); System.out.print(staticVar); } } </pre> <p>以下代码正确的是:</p> <p>A. 能正常运行, 并且输出 132ABC</p> <p>B. 能正常运行, 并且输出 123CBA</p> <p>C. 编译失败</p>	<p>类方法只能访问类变量, 对象方法也可以访问类变量</p>
3	<pre> public class TT { private double d; private String str; private char chr ; public static void main(String args[]) { TT tt = new TT(); System.out.println(tt.chr + tt.d + tt.str); } } </pre> <p>关于以上代码, 说法正确的是: ()</p> <p>A. 运行时抛异常</p> <p>B. 输出 " 0.0null "</p>	<p>变量的默认值</p>

	<p>C. 输出” \u00000.0null”</p> <p>D. 输出” 00.0null”</p>	
4	<p>下列声明正确的是: ()</p> <p>A. short myshort = 99S;</p> <p>B. String name = ‘Excellent tutorial Mr Green’ ;</p> <p>C. char c = 17c;</p> <p>D. int z = 015;</p>	<p>基本类型的赋值 不存在字母 c 和 s 这种文字指示符, 一个 String 必须包含在双引号中, 而不是例子中的单引号。</p>
5	<p>下列关于变量的初始化说法正确的是: ()</p> <p>A. 静态变量必须显式初始化才能使用</p> <p>B. 实例变量必须显式初始化才能使用</p> <p>C. 局部变量必须显式初始化才能使用</p> <p>D. 以上说法都正确</p>	<p>局部变量没有默认值</p>
6	<p>请阅读代码:</p> <pre>public class Test { public static void main(String args[]) { final int len = 5; int array[] =new int[len]; for(int i = 0; i < len - 2; i++) { array[i] = i; } for(int i = 0; i < array.length; i++) { System.out.print(array[i] + ","); } } }</pre> <p>以上代码下列说法正确的是: ()</p> <p>A. 输出 "0,1,2,0,0,"</p> <p>B. 输出 "0,1,2,"</p> <p>C. 运行时抛异常</p>	<p>刚构造的数组中的元素总是设为默认值</p>

7	<p>下面几种声明数组的方法，错误的有（）</p> <p>A. <code>char chr[] = new char[3]{'A','B','C'};</code></p> <p>B. <code>char chr[] = new char[3];</code></p> <p>C. <code>char chr[] = {'A','B','C'};</code></p> <p>D. <code>char [][]chr = new char[10][];</code></p>	数组的声明
8	<p>下面两种数组的拷贝，哪个更有效率？（）</p> <p><code>String[] from = new String[10];</code> <code>String to = new String[10];</code></p> <p>A. <code>for (int i = 0, n = from.length; i < n; i++)</code> <code>{</code> <code>to[i] = from[i];</code> <code>}</code></p> <p>B. <code>System.arraycopy(from, 0, to, 0, from.length);</code></p>	拷贝数组应当使用 System.arraycopy() 函数 以提高效率
9	<p>下面关于 final 变量说法正确的是:()</p> <p>A. 声明为 final 的数组其元素只能赋值一次;</p> <p>B. 声明为 final 的整型变量其内容不能被改变;</p> <p>C. 声明为 final 的 String 对象可以通过 replace 函数改变其内容</p>	一个 final 变量的值 不能被改变，并且必 须在一定的时刻赋值
10	<p>当你试着编译运行下面的代码的时候，可能会发生什么？</p> <pre>class Base{ public final void amethod() { System.out.println("amethod"); } } public class Fin extends Base { public static void main(String argv[]) { Base b = new Base(); b.amethod(); } }</pre> <p>A. 编译失败，包含 final 方法的类，也必须声明为 final</p> <p>B. 编译失败，包含 final 方法的类，不能被继承</p> <p>C. 运行期异常，因为 Base 类为被声明为 final</p> <p>D. 运行成功，并且输出"amethod"</p>	final 的基本用法

11	<p>以下说法正确的是: ()</p> <p>A. 只要是用 final static int 修饰的数据都恒定不变的, 多次重新启动虚拟机反复运行后, 它的值还是恒定不变的</p> <p>B. 只要是用 final static int 修饰的数据都可以用在 switch - case 语句的 case 子句中</p> <p>C. 只能用 final static 定义常量</p> <p>D. 保留字 const 不能用来定义常量</p>	常量
12	<p>以下程序输出什么?</p> <pre> class Parent { protected String pStr; public Parent() { invokePrint(); pStr = "parent"; } protected void invokePrint() { System.out.println("Parent print, pStr = " + pStr); } } public class Child extends Parent { private String cStr; public Child() { super(); cStr = "child"; } protected void invokePrint() { System.out.println("Child print, pStr = " + pStr + " cStr = " + cStr); } public static void main(String[] args) </pre>	在构造过程中避免调用子类可以覆写的函数

	<pre> { Child c = new Child(); c.invokePrint(); Parent p = new Parent(); p.invokePrint(); } } </pre> <p>A.</p> <p>Child print, pStr = null cStr = null Child print, pStr = parent cStr = child Parent print, pStr = null Parent print, pStr = parent</p> <p>B.</p> <p>Parent print, pStr = null Parent print, pStr = parent Parent print, pStr = null Parent print, pStr = parent</p> <p>C.</p> <p>Child print, pStr = parent cStr = null Child print, pStr = parent cStr = child Parent print, pStr = null Parent print, pStr = parent</p> <p>D.</p> <p>Child print, pStr = null cStr = null Child print, pStr = null cStr = child Parent print, pStr = parent Parent print, pStr = parent</p>	
13	<p>运行下面程序输出的是:</p> <pre> class Parent { protected static String str = getParentMessage(); static { str = "static block init"; } } </pre>	<p>考察类的初始化过程，需要注意的是静态内部类的初始化和正常声明类的初始化一样：需要用时才加载初始化</p>

	<pre> private static String getParentMessage() { System.out.println("getParentMessage"); return "getParentMessage"; } static class InnerClass { static { str = "inner class static block init"; } } public class Child extends Parent { Child() { System.out.println("final str=" + str); } public static void main(String args[]) { new Child(); } } </pre> <p>A. <code>getParentMessage</code> <code>final str=static block init</code></p> <p>B. <code>getParentMessage</code> <code>final str=inner class static block init</code></p> <p>C. 抛出运行时异常</p> <p>D. <code>final str=static block init</code></p>	
14	<p>下列关于构造函数说法正确的是: ()</p> <p>A. 构造函数不能被重载</p> <p>B. 构造函数不能被覆写</p> <p>C. 构造函数可以返回一个对象的引用</p> <p>D. 构造函数代码执行顺序是从当前的类层级到它祖先的类</p> <p>E. 构造函数不能抛异常</p>	构造函数

第 4 章 运算

编号	试 题	知识点（考察点）说明
1	<p>在下面给定的类中，哪一个会编译出错？</p> <pre> interface IFace{} class CFace implements IFace{} class Base{} public class ObRef extends Base{ public static void main(String argv[]){ ObRef ob = new ObRef(); Base b = new Base(); Object o1 = new Object(); IFace o2 = new CFace(); } } </pre> <p>A. o1=o2; B. b=ob; C. ob=b; D. o1=b;</p>	<p>将一个对象引用赋值给另一个的一般规则是，你可以对继承树向上赋值但不能向下赋值。可以这样想，如果将一个子类的实例赋值给基类，Java 知道哪个方法会在子类中。但是一个子类可能有基类没有的额外方法。你可以使用操作符强制转换。</p>
2	<p>下面哪一行可以编译成功？</p> <p>A. int i = 10; B. float f = 1.3; C. char c = "c"; D. byte b = 257;</p>	<p>整数的默认类型是 int，构造长整形变量时，需要加上后缀 L，避免溢出。</p> <p>浮点数的默认类型是 double，将浮点数赋值给值域比它小的变量时，需要进行强制类型转换；构造单精度浮点数时，可以用后缀 f 指定。</p>
3	<p>编译，运行下面代码段会发生什么情况？</p> <pre> 1. byte b1 = 1; 2. byte b2 = 127; 3. short s = b1 + b2; </pre> <p>A.第 1 行编译错误 B.第 2 行编译错误 C.第 3 行编译错误 D.可以正常编译</p>	<p>值域小的向值域大的方向转换,值域大的向值域小的方向转换需要进行强制类型转换</p>

4	<p>尝试编译运行以下代码时会发生什么情况？</p> <pre> public static void main(String[] args) { boolean b1 = false; if(b1 && secondExp()) { System.out.print("true"); } else { System.out.print("false"); } } private static boolean secondExp() { System.out.print("secondExp"); return true; } </pre> <p>A. 编译错误 B. 输出 secondExp, true C. 输出 secondExp, false D. 输出 false</p>	<p>对于逻辑运算符“&&”来说，如果第一个操作数为假，第二个操作数的值就没有作用了，也不会执行，所有的结果都为假。</p> <p>对于逻辑运算符“ ”来说，如果第一个操作数为真，第二个操作数的值就没有作用了，也不会执行，所有的结果都为真</p>
5	<p>以下所给代码段的输出是什么？</p> <pre> class Father{} public class Child extends Father { public static void main(String[] args) { Child child = new Child(); Father father = new Father(); System.out.println(child.getClass() == father.getClass()); System.out.println(child instanceof Father); } } </pre> <p>A. 输出 true,false B. 输出 true,true C. 输出 false,true D. 输出 false,false</p>	<p>getClass()用于获取对象的真实类型，两个有继承关系的类分别调用getClass()获得的类是不同的。</p> <p>instanceof 用来判断一个对象是否是某种类型的，即 IS-A 的关系。子类对象是父类对象的实例。</p>

6	<p>以下所给代码段的输出是什么?</p> <pre>System.out.println(null instanceof Object);</pre> <p>A. Object B. true C. false D. null</p>	<p>null 对象不是任何类的实例, 所有 instanceof 运算符返回值总是 false</p>
7	<p>以下程序输出结果是什么?</p> <pre>public static void main(String[] args) { Point p1 = new Point(0, 0); Point p2 = new Point(0, 0); modifyPoint(p1, p2); System.out.println "[" + p1.x + "," + p1.y + "], [" + p2.x + "," + p2.y + "]); } private static void modifyPoint(Point p1, Point p2) { Point tmpPoint = p1; p1 = p2; p2 = tmpPoint; p1.setLocation(5, 5); p2 = new Point(5, 5); } A. [0,0], [0,0] B. [5,5], [0,0] C. [0,0], [5,5] D. [5,5], [5,5]</pre>	<p>对象是通过引用传递的</p>

8	<p>以下所给代码段的输出是什么?</p> <pre>class ValHold { public int i = 10; } public class ObParm { public static void main(String argv[]) { ObParm o = new ObParm(); o.amethod(); public void amethod() { int i = 99; ValHold v = new ValHold(); v.i=30; another(v,i); System.out.print(v.i); } public void another(ValHold v, int i) { i=0; v.i = 20; ValHold vh = new ValHold(); v = vh; System.out.print(v.i+ ", "+i+", "); } } }</pre> <p>A. 10,0, 30 B. 20,0,30 C. 20,99,30 D. 10,0,20</p>	<p>所有的变量（基本类型值和对象的引用值）都是值传递。但是这并不是全部情况，对象是经常通过 Java 的引用变量来操作的。所以也可以说对象是通过引用传递的（引用变量通过值传递）。调用者对基本类型参数（int，char 等）的拷贝在相应参数变化时不会改变。但是，在被调用方法改变相应作为参数传递的对象（引用）字段时，调用者的对象也改变其字段。当你方法传递基本类型参数，是直接传递值。方法得到它的拷贝，任何修改都不会在外部方法得到反映</p>
---	--	--

9	<p>以下程序输出结果是什么?</p> <pre>String strTest1 = "abc"; String strTest2 = new String("test"); if (strTest1 == "abc") { System.out.print("true"); } else { System.out.print("false"); } if (strTest2 == "test") { System.out.print("true"); } else { System.out.print("false"); }</pre> <p>A. true>true B. true>false C. false>true D. false>false</p>	<p>对于对象, “==” 运算的语义是用来判断对象是不是同一个对象</p>
10	<p>以下程序输出结果是什么?</p> <pre>String strTest1 = "abc"; String strTest2 = new String("test"); if (strTest1.equals("abc")) { System.out.print("true"); } else { System.out.print("false"); } if (strTest2.equals("test")) { System.out.print("true"); } else { System.out.print("false"); }</pre> <p>A. true>true B. true>false C. false>true D. false>false</p>	<p>equals() 函数用来判断对象的内容(或者是属性)是否相同</p>

11	<p>下面哪种说法是正确的?</p> <pre>public class CompObj { public int x; public int y; public boolean equals(Object o1) { CompObj compO1 = (CompObj)o1; return this.x == compO1.x; } }</pre> <p>A. 该 equals 函数实现错误, 没有判断是不是相同类型的对象, 并且在 o1 是 null 的时候会抛出异常</p> <p>B. 该 equals 函数实现正确</p> <p>C. 该 equals 函数实现错误, 没有比较 y 的值</p>	覆写 equals() 的注意事项
12	<p>下面哪种说法是正确的?</p> <pre>int i = 0; public int hashCode() { return i++; }</pre> <p>A. 该 hashCode 函数实现错误, 相同的对象调用多次 hashCode() 返回不同的结果</p> <p>B. 该 hashCode 函数实现正确</p>	覆写 hashCode() 函数的注意事项

	<p>以下类哪个是安全拷贝的: ()</p> <pre>class A implements Cloneable { private int a = 1; private boolean b = false; public Object clone() { try { return (super.clone()); } catch(CloneNotSupportedException cnes) { throw new InternalError(); } } }</pre>	
13	<pre>class B implements Cloneable { private int a = 1; private boolean b = false; private Hashtable h = new Hashtable(); public Object clone() { try { return (super.clone()); } catch(CloneNotSupportedException cnes) { throw new InternalError(); } } }</pre> <p>A. 类 A B. 类 B C. 类 A 和类 B 都不是安全拷贝</p>	<p>为防止对象被意外修改, 应当在必要时拷贝对象</p>

14	<p>下面代码段中深度克隆的实现方式是否正确?</p> <pre>public class MyCloneableClass { private Map map = new HashMap(); public Object clone() throws CloneNotSupportedException { return super.clone(); } }</pre> <p>A. 不正确, 没有对 map 进行深度克隆 B. 不正确, 既没有实现 Cloneable 接口, 又没有对 map 进行深度克隆 C. 正确, 可以完成深度克隆工作 D. 不正确, 没有实现 Cloneable 接口</p>	<p>如何正确覆写 clone() 函数</p>
15	<p>以下代码的输出结果是:</p> <pre>String abc = "abc"; abc.replace('b', 'd'); System.out.println(abc);</pre> <p>A. abc B. adc C. 答案 A 和 B 都不对</p>	<p>String 对象一旦被创建, 它就决不能被改变</p>
16	<p>不考虑上下文, 以下创建了几个对象? 并且指出输出结果</p> <pre>String s1 = new String("abc"); String s2 = new String("abc"); if(s1== s2) { System.out.println("s1,s2 refer to the same object"); }</pre> <p>A. 2; "s1,s2 refer to the same object" B. 3; 无输出 C. 4; "s1,s2 refer to the same object" D. 5; 无输出</p>	<p>本考点是字符串的创建。由于字符串对象的大量使用 (它是一个对象, 一般而言对象总是在 heap 分配内存), java 中为了节省内存空间和运行时间 (如比较字符串时, “==” 比 equals() 快), 在编译阶段就把所有的字符串文字放到一个文字池 (pool of literal) 而运行时文字池成为常量池的一部分。文字池的好处, 就是该池中的所用相同的字符串常量被合并, 只占用一个空间。对两个引用变量, 使用 “==” 判断它们的值 (引用) 是否相同, 即指向同一个对象</p>

17	<p>下面程序输出的结果:</p> <pre>byte[]a = {49,50,51,52}; System.out.println(a.toString());</pre> <p>A. "1234" B. 49 C. 49505152 D. 以上都不是</p>	<p>byte[] 数组的 toString() 函数 实际上是 Object 的 toString 函数, 而不是数组的内容。想通过 byte[]数组的构造字符串, 需要调用 new String(byte[])构造函数。</p>
18	<p>下面说法错误的是:</p> <p>A. StringBuffer 的 append() 效率好于" + " ;</p> <p>B. 多次循环内使用 StringBuffer 的 append()的效率要好与" + " ;</p> <p>C. String s = "s1" + "s2" ;和 String s = new StringBuffer().append("s1").append("s2") 效率一样。</p> <p>D. 在声明 StringBuffer 对象时, 指定合适的 capacity, 不要使用默认值 (16) 。</p>	<p>简单的认为 StringBuffer 的 append() 效率好与" + " 是错误的, 需要了解 "+"的工作原理。</p>
19	<p>下面的代码是否存在内存浪费:</p> <pre>String tmp = ""; for(int i=0; i<9999; i++) { tmp+="x"; //do something }</pre> <p>A. 存在 B. 不存在</p>	<p>String 是非可变性的, 所以, 当执行 tmp+="x", 实际上是另外创建了一个对象, 而原来的对象成为了垃圾(当它没有其他引用的时候), 这样的话一个循环就会产生 n 个对象, 从而造成内存浪费</p>

第 5 章 垃圾收集

编号	试 题	知识点 (考察点) 说明
1	<p>下述代码中, 执行完语句 1 后, b 是否会被回收?</p> <pre>import java.util.*; public class Stack { private Object[] elements; private int size = 0; public Stack(int initialCapacity)</pre>	<p>当对象不可达的时候, 这个对象就成为了可回收的</p>

```
{
    this.elements = new Object[initialCapacity];
}
public void push(Object e)
{
    ensureCapacity();
    elements[size++] = e;
}
public Object pop()
{
    if (size == 0)
        throw new EmptyStackException();
    return elements[--size];
}
private void ensureCapacity()
{
    if (elements.length == size)
    {
        Object[] oldElements = elements;
        elements = new Object[2 *
elements.length + 1];
        System.arraycopy(oldElements, 0,
elements, 0, size);
    }
}

public static void main(String[] args)
{
    Stack stack=new Stack(10);
    Integer i=new Integer(10);
    stack.push(i);
    Object b=stack.pop();
    b=null;  ////////////语句 1
}
}
```

- A. 可以
B. 不可以

2	<p>请阅读以下代码:</p> <pre>01. class RJMould 02. { 03. StringBuffer sb; 04. 05. public static void main(String argv[]) 06. { 07. RJMould rjm = new RJMould(); 08. rjm.kansas(); 09. } 10. 11. public void kansas() 12. { 13. sb = new StringBuffer("Manchester"); 14. StringBuffer sb2 = sb; 15. StringBuffer sb3 = new StringBuffer("Chester"); 16. sb = sb3; 17. sb3 = null; 18. sb2 = null; 19. } 20. }</pre> <p>执行哪一行代码以后, 第 13 行创建的 StringBuffer 对象成为可回收的对象?</p> <p>A. 14 B. 15 C. 16 D. 18</p>	<p>当对象不可达的时候, 这个对象就成为了可回收的</p>
---	--	--------------------------------

3	<p>请阅读类 Stack 代码，存在内存泄露隐患的方法是()</p> <pre>import java.util.*; public class Stack { private Object[] elements; private int size = 0; public Stack(int initialCapacity) { this.elements = new Object[initialCapacity]; } public void push(Object e) { ensureCapacity(); elements[size++] = e; } public Object pop() { if (size == 0) throw new EmptyStackException(); return elements[--size]; } private void ensureCapacity() { if (elements.length == size) { Object[] oldElements = elements; elements = new Object[2 * elements.length + 1]; System.arraycopy(oldElements, 0, elements, 0, size); } } }</pre> <p>A. pop() B. push() C. ensureCapacity() D. 没有隐患</p>	<p>当对象不可达的时候，这个对象就成为了可回收的。</p>
---	---	--------------------------------

第21页, 共52页

第 6 章 流程控制和差错处理

编号	试 题	知识点（考察点）说明
1	<p>下列关于 switch 语句的 default 分支说法正确的是（ ）</p> <p>A. switch 语句中必须有 default 分支</p> <p>B. switch 语句中的 default 分支不是必须的，所以在编码中不考虑 default 分支也不会有问题</p> <p>C. 从语法上看, switch 语句中的 default 分支不是必须的, 但在编码中不考虑 default 分支有可能发生问题</p>	switch 语句必须的配套的 default 分支
2	<p>下列程序的输出结果是</p> <pre>int a = 2; switch(a) { case 1: System.out.println("The input is 1"); case 2: System.out.println("The input is 2"); case 3: System.out.println("The input is 3"); default: System.out.println("Bad input"); }</pre> <p>A. The input is 2 Bad input</p> <p>B. The input is 2 The input is 3 Bad input</p> <p>C. The input is 2</p> <p>D. Bad input</p>	各个分支之间不能遗漏 break
3	<p>语句 switch(expr){...} , 合法的表达式 expr 可以具有哪些类型</p> <p>A. long</p> <p>B. string</p> <p>C. unsigned int</p> <p>D. byte、char、short、int</p>	switch 语句只能用 byte, char, short 或者 int 类型作参数



4	<p>下列代码输出结果为? ()</p> <pre>int i = 1; int i2 = 0; switch (i) { case i2: System.out.print("A,"); case 1: System.out.print ("B,"); case 2: System.out.print ("C,"); default: System.out.print ("ABC"); }</pre> <p>A. B, B. B,C C. B,C,ABC D. 程序编译不过</p>	<p>case 的表达式必须是常量，不能是变量等。switch 能将整数选择因子的结果与每个整数值比较。若发现相符的，就执行对应的语句（简单或复合语句）。若没有发现相符的，就执行 default 语句。大家会注意到 case 均可以以一个 break 结尾。这样可使执行流程跳转至 switch 主体的末尾。这是构建 switch 语句的一种传统方式，但 break 是可选的。若省略 break，会继续执行后面的 case 语句的代码，直到遇到一个 break 为止。</p>
5	<p>关于 try block，说法正确的是 () ?</p> <p>A. 每个 try block 必须有一个以上的 catch block 相对应 B. 每个 try block 必须有一个 finally block 相对应 C. 每个 try block 至少需要有一个 finally block 或者一个 catch block 相对应 D. 每个 try block 至少需要有一个 finally block 和一个 catch block 相对应</p>	<p>异常处理的基本语法，一个 try 只需要有 catch 或者 finally 对应即可。</p>
6	<p>下面的代码中，输入流 reader 的关闭放在什么位置是最合适的 ()</p> <pre>public static void readFileByLines(String fileName) throws IOException { File file = new File(fileName); BufferedReader reader = null; try { reader = new BufferedReader(new FileReader(file)); String tempString = null; int line = 1; while((tempString = reader.readLine()) != null) { line++; } } }</pre>	<p>为保证 IO 流的关闭能够在任何情况下都能够被执行，IO 流的关闭放在 finally 中进行。</p>

	<pre> reader.close(); //////////// 答案 A } catch(Exception e) { reader.close(); //////////// 答案 B } finally { if(reader != null) { try { reader.close(); //////////// 答案 C } catch(IOException e1) { e1.printStackTrace(); } } } } </pre>	
7	<p>如下代码的正常输出是什么? (请注意输出的正常顺序)</p> <pre> public class Test { public static void main(String[] args) { System.out.println(t()); } public static int t() { int b = 23; try { System.out.println("Begin"); return b = 88; }catch(Exception e) { System.out.println("error :" + e); } finally { if(b > 25) </pre>	<p>如果一个 return 语句被嵌入 try 代码块中, 那么, finally 代码块中的代码应在 return 前执行</p>

	<pre> { System.out.println("b > 25 : " + b); b = 66; } System.out.println("finally"); } return b; } } </pre> <p>A.</p> <pre> Begin b > 25 : 88 finally 88 </pre> <p>B.</p> <pre> Begin b > 25 : 88 finally 66 </pre> <p>C.</p> <pre> Begin 88 </pre> <p>D.</p> <pre> Begin 66 </pre>	
8	<p>有以下代码，正确的选项是：（ ）</p> <pre> 01 class ExceptionBase extends Exception{} 02 03 class ExceptionSub1 extends ExceptionBase{} 04 05 class Base 06 { 07 void method() throws ExceptionSub1{} 08 } 09 10 class Sub1 extends Base 11 { 12 void method() throws ExceptionBase{} 13 } 14 15 class Sub2 extends Base 16 { </pre>	<p>子类的覆写方法只能抛出父类中声明过的异常或者异常的子类</p>

	<pre> 17 void method() throws ExceptionSub1{ 18 } </pre> <p>A. 可以正常编译运行 B. 第 7 行编译出错 C. 第 12 行编译出错 D. 第 17 行编译出错</p>	
9	<p>阅读以下程序，如果要删除容器中所有"two",哪一段代码填入 deleteElement()方法中是正确的 ()</p> <pre> public static void main(String[] args) { List list = new ArrayList(); list.add("one"); list.add("two"); list.add("two"); list.add("two"); deleteElement(list); } </pre> <pre> private static void deleteElement(List list) { //在这里填入一段代码 } </pre> <p>代码一:</p> <pre> for(int i = 0; i < list.size(); i++) { String item = (String)list.get(i); if(item.equals("two")) { list.remove(i); } } </pre>	避免在循环中删除元素时循环次数错误

```
}
```

代码二:

```
for(Iterator iter = list.iterator(); iter.hasNext(); )
{
    String item = (String)iter.next();
    if(item.equals("two"))
    {
        iter.remove();
    }
}
```

代码三:

```
int i = 0;
for(Iterator iter = list.iterator(); iter.hasNext(); )
{
    String item = (String)iter.next();
    if(item.equals("two"))
    {
        list.remove(i++);
    }
}
```

代码四:

```
for(int i = 0, len = list.size(); i < len; i++)
{
    String item = (String)list.get(i);
    if(item.equals("two"))
    {
        list.remove(i);
    }
}
```

A. 代码一 B. 代码二 C. 代码三 D. 代码四

	<p>下列哪种说法正确: ()</p> <pre>private static void testA() { String[] strArr = {"33", "44", "55", "66"}; try { for(int i = 0; i < strArr.length; i++) { int val = Integer.parseInt(strArr[i]); } } catch(Exception ex) { ex.printStackTrace(); } }</pre>	
10	<pre>private static void testB() { String[] strArr = {"33", "44", "55", "66"}; for(int i = 0; i < strArr.length; i++) { try { int val = Integer.parseInt(strArr[i]); } catch(Exception ex) { ex.printStackTrace(); } } }</pre>	循环的效率
	<p>A. 方法 testA()效率比 testB()高</p> <p>B. 方法 testA()效率比 testB()低</p> <p>C. 方法 testA()效率与 testB()一样</p> <p>D. 无法确定</p>	

第 7 章 访问控制、继承和多态

编号	试 题	知识点（考察点）说明
1	<p>对于类 B 的说法不正确的是 ()</p> <pre>public class B { private A aa = null; public A getA() { return aa; } public void setA(A aa) { this.aa = aa; } public void doTask() { aa.method(); } }</pre> <p>A. 类 B 的设计具有较好的封装性 B. getA()方法能返回任意 A 的子类型 C. setA()方法能传入任意 A 的子类型 D. 变量名 aa 的更改会影响到客户</p>	<p>实例变量的声明和获取 尽量避免将一个类中的各变量都声明为 public，建议通过 getter/setter 访问。</p>
2	<p>关于以下的代码片断，说法错误的是? ()</p> <pre>public class A { int a; protected final int b = 1; //... }</pre> <p>A. a 对于与 A 在同一个包的类是可见的 B. a 对于 A 的子类是可见的 C. b 对于与 A 在同一个包的类是可见的 D. b 对于 A 的子类是可见的</p>	<p>类、变量、函数的访问权限</p>

3	<p>对于下面代码片断说法正确的是 ()</p> <pre>public interface S { int a = 0; void method(); }</pre> <p>A. 此代码编译不能通过, 因为 method () 没有 abstract 修饰 B. 此代码编译不能通过, 因为 interface 中不能给变量赋值 C. 变量 a 的访问权限是包访问权限 D. 变量 a 的访问权限是 public 的</p>	<p>接口变量和方法的默认访问权限 接口的属性默认为 public、final 的; 接口的方法定义默认为 public abstract 的;</p>
4	<p>对于下面代码片断说法正确的是 ()</p> <pre>public class OuterClass { int a; static int b; private void method() { System.out.println("method()"); } class Inner { static int c;//A void innerMethod() { method();//B } } public static void main(String[] s) { Inner inner = new Inner();//C inner.innerMethod();//D } }</pre> <p>A. A 处编译错误, 因为只有静态内部类才能声明静态成员 B. B 处编译错误, 因为 method() 是 private 的 C. 此类编译正确</p>	<p>内部类对包含它的外部类中的字段, 方法的访问权限 考点说明 内部类是在一个类的内部嵌套定义的类, 它可以是其它类的成员, 也可以在一个语句块的内部定义, 还可以在表达式内部匿名定义。 内部类有如下特性: 可以使用包含它的类的静态和实例成员变量, 即使它们在外围类中是 private 的。 若被声明为 static, 就变成了顶层类, 不能再访问其外部类的非静态成员。 若想在 Inner Class 中声明任何 static 成员, 则该 Inner Class 必须声明为 static。</p>
5	<p>类 Test 与 Outer 位于同一个包, Inner 为 Outer 的非静态内部类, 那么在类 Test 中如何创建 Inner 的实例?</p> <p>A. Outer.Inner i = new Outer ().new Inner (); B. Outer.Inner i = Outer ().new Inner (); C. Outer.Inner i = Outer ().new new Inner ();</p>	<p>内部类对包含它的外部类中的字段, 方法的访问权限 考点说明 内部类是在一个类的内部嵌套定义的类, 它可以是其它类的成员, 也</p>

	D. Outer i = Outer.new ().Inner ();	<p>可以在一个语句块的内部定义，还可以在表达式内部匿名定义。</p> <p>内部类有如下特性：</p> <p>可以使用包含它的类的静态和实例成员变量，即使它们在外围类中是 private 的。</p> <p>若被声明为 static,就变成了顶层类,不能再访问其外部类的非静态成员。</p> <p>若想在 Inner Class 中声明任何 static 成员,则该 Inner Class 必须声明为 static。</p>
6	<p>对下面代码片断说法正确的是 ()</p> <pre> public class OuterClass { private int a; private void method() { int methodVal = 3; class TempInner { int cccc = methodVal; //A Inner inner = new Inner(); //B void doTest() { inner.c = 5; //C } } } class Inner { int c = a; void innerMethod() { method(); } } } </pre> <p>A. A 处编译错误，因为内部类只能访问 final 的局部变量</p> <p>B. B 处编译错误，因为非静态内部类实例化时需要从属于一个外部类对象。</p>	<p>内部类可以访问声明为 final 的局部变量(常用的)</p> <p>考点说明</p> <p>内部类可以使用外围类的静态和实例成员变量，即使它们在外围类中是 private 的。</p> <p>但是对于局部变量，只有 final 的才能被内部类访问。</p>

	<p>C. C 处编译错误</p> <p>D. 此类编译正确</p>	
7	<p>下面关于继承的哪些叙述是正确的。</p> <p>A. 在 java 中只允许单一继承。</p> <p>B. 在 java 中一个类只能实现一个接口。</p> <p>C. 在 java 中一个类不能同时继承一个类和实现一个接口。</p> <p>D. java 中的一个接口可同时继承多个接口和一个父类。</p>	<p>类的单继承性</p> <p>考点说明</p> <p>Java 中所有的类都是通过直接或间接地继承 java.lang.Object 类得到的。继承而得到的类称为子类，被继承的类称为父类。子类不能继承父类中访问权限为 private 的成员变量和方法。子类可以重写父类的方法，及命名与父类同名的成员变量。但 Java 不支持多重继承，即一个类从多个超类派生的能力。</p>
8	<p>下列描述正确的是 ()</p> <p>A. 一个接口可以继承多个接口</p> <p>B. 一个 Java 类可以实现多个接口，或者继承多个类</p> <p>C. 一个接口可以继承一个类，但是可以继承多个接口</p> <p>D. 上面说法都正确</p>	<p>接口的多继承性</p> <p>Java 只允许类的单继承，但可以通过接口实现多继承。</p>
9	<p>对于下列代码片断，描述正确的是 ()</p> <pre> class Base { public Base () { System.out.println("Base()"); } public Base (int i) { System.out.println("Base(int i)"); } } public class MyOver extends Base { public void MyOver(int i) { } public static void main (String argv []) { MyOver m = new MyOver (10); } } </pre>	<p>构造函数</p> <ol style="list-style-type: none"> 1、构造函数与类名必须相同，且不能有返回值。 2、默认情况下，编译器会为每个类默认生成一个无参构造函数。 3、但是若类中定义了构造函数，那么编译器就不会自动生成默认构造函数。 4、子类的构造函数中若没显式调用父类构造函数，那么会编译器会自动调用父类的默认构造函数，若此时父类没有默认构造函数，那么将会编译错误。

	<p>A. 运行此程序输出 “Base(int i)”</p> <p>B. 运行此程序输出 “Base()”</p> <p>C. 此程序运行后什么也不输出，因为 MyOver 的构造函数中没有显示调用父类构造函数</p> <p>D. 此程序编译错误</p>	
10	<p>在 Java1.4 版本，关于 Overriding, 说法正确的是? ()</p> <p>A. 覆写方法的方法名和参数列表必须和被覆写方法一致，而返回值类型不必一致</p> <p>B. 覆写方法的方法名，返回值类型和参数列表必须和被覆写方法一致</p> <p>C. 覆写方法的方法名和参数列表必须和被覆写方法一致，而返回值类型可以是被覆写方法返回值类型的子类</p> <p>D. 覆写方法的方法名和返回值类型必须和被覆写方法一致，而参数类型可以是被覆写方法参数类型的子类</p>	<p>函数的重载和覆写以及两个的区别</p> <p>考点说明</p> <p>成员函数重载，即几个名字相同、参数不同的成员函数。</p> <p>成员函数覆写，即子类的成员函数与父类的成员函数名字，参数，返回值都相同时，子类成员函数将覆写父类成员函数的行为。</p>
11	<p>对于下面的 Upton 类</p> <pre>public class Upton{ public static void main(String argv[]){ public void amethod(int i){} //Here } }</pre> <p>下面哪一个在替换//Here 后是不合法的?</p> <p>A. public int amethod(int z){}</p> <p>B. public int amethod(int i,int j){return 99;}</p> <p>C. protected void amethod(long l){ }</p> <p>D. private void anothermethod(){}</p>	<p>方法的重载和覆写以及两个的区别</p> <p>考点说明</p> <p>成员函数重载，即几个名字相同、参数不同的成员函数。</p> <p>成员函数覆写，即子类的成员函数与父类的成员函数名字，参数，返回值都相同时，子类成员函数将覆写父类成员函数的行为。</p>
12	<p>考虑下面的两个类 ClassA、ClassB</p> <pre>public class ClassA { public void methodOne(int i) {} public void methodTwo(int i) {} public static void methodThree(int i) {} public static void methodFour(int i) {} } public class ClassB extends ClassA { public static void methodOne(int i) {} public void methodTwo(int i) {} public void methodThree(int i) {} public static void methodFour(int i) {} }</pre>	<p>static 函数不能被覆写，所以 static 函数是没有多态性的</p> <p>Static 方法是属于类的，它不能被覆写，不具多态性。</p>

	<p>以下说法正确的是？ ()</p> <p>A. 子类 ClassB 中 methodOne 方法 overrides 父类 ClassA 的方法</p> <p>B. 子类 ClassB 中 methodTwo 方法 overrides 父类 ClassA 的方法</p> <p>C. 子类 ClassB 中 methodThree 方法 overrides 父类 ClassA 的方法</p> <p>D. 子类 ClassB 中 methodFour 方法 overrides 父类 ClassA 的方法</p>	
13	<p>下面说法正确的是 ()</p> <pre> class Base { abstract public void myfunc(); public void another() { System.out.println("Another method"); } } public class Abs extends Base { public static void main(String argv[]) { Abs a = new Abs(); a.amethod(); } public void myfunc() { System.out.println("My func"); } public void amethod() { myfunc(); } } </pre> <p>A. 代码编译通过且输出 "My Func".</p> <p>B. 编译错误，因为 Base 没有被声明为 abstract。</p> <p>C. 编译正确，运行时错误。</p> <p>D. 编译器将报错，因为类 Base 中的方法 myfunc()没有方法体。</p>	<p>抽象类与接口</p> <p>考点说明</p> <ol style="list-style-type: none"> 1、抽象类不能直接初始化， 2、抽象类可以没有抽象方法， 3、有抽象方法的类必须是抽象类或者接口 4、接口中不能有方法实现 <p>抽象类必须被继承，抽象方法必须被重写。抽象方法只需声明，无需实现；抽象类不能被实例化，抽象类不一定要包含抽象方法。若类中包含了抽象方法，则该类必须被定义为抽象类。</p>

14	<p>下列说法不正确的是? ()</p> <p>A. 面向接口编程即要求我们在设计时只用 interface 不用 class</p> <p>B. 面向接口编程与面向实现编程是相对的, 即在设计时应当尽量依赖于抽象的接口, 而不是具体的实现。</p> <p>C. 当一函数声明的返回值为类型 A 时, 那么事实上它可以返回任意类型 A 的子类型对象</p> <p>D. 当一变量声明为类型 B 时, 那么事实上它可以被赋值为任意类型 B 的子类型对象</p>	<p>面向接口和抽象编程</p> <p>面向接口编程是设计可扩展性框架的必要条件, 它要求我们:</p> <ol style="list-style-type: none"> 1、函数的参数和返回值尽量使用接口。 2、定义变量时尽量使用接口。
15	<p>关于静态内部类, 说法正确的是? ()</p> <p>A. 静态内部类持有其外部类对象的引用, 因此它不能访问其外部类的所有成员</p> <p>B. 静态内部类持有其外部类对象的引用, 因此它能访问其外部类的所有成员</p> <p>C. 静态内部类在实例化时需要有其外围类的对象。</p> <p>D. 当内部类无需访问其外部类成员时, 应当尽量使用静态内部类</p>	<p>静态内部类和非静态内部类的区别 (静态内部类不持有外部类的引用)</p> <ol style="list-style-type: none"> 1、非静态内部类都持有其外部类对象的引用, 可以访问其外部类的所有成员, 2、而静态内部类则不持有其外部类对象的引用, 只能访问其外部类的静态成员。

第 8 章 并发与多线程

编号	试 题	知识点 (考察点) 说明
1	<p>对于 RunHandler 类来说, 以下哪些选项必须为真时才能使得 runHandle 的实例能够用于编写如下代码?()</p> <pre>class Test { public static void main(String[] args) { Thread t = new Thread(new RunHandle()); t.start(); } }</pre> <p>A. RunHandler 必须实现 java.lang.Runnable 接口</p> <p>B. RunHandler 必须扩展 Thread 类</p> <p>C. RunHandler 必须提供一个声明为 public,并且返回 void 的 run()方法</p> <p>D. RunHandler 必须提供一个 init()方法</p>	<p>虽然可以通过扩展 Thread 来表明实现 Runnable 接口, 但不是必须扩展 Thread 来实现, 所以 B 错误。</p>

2	<p>一个程序消耗较多的 CPU 资源, 多启动几个线程就可以改善性能。()</p> <p>A. 正确</p> <p>B. 错误</p>	<p>线程的基本概念,对消耗 CPU 资源的计算, 多启动线程也没有任何帮助</p>
3	<p>下列哪一项是让线程开始运行的正确代码?</p> <p>A.</p> <pre>public class TStart extends Thread{ public static void main(String argv[]){ TStart ts = new TStart(); ts.start(); } public void run(){ System.out.println("Thread starting"); } }</pre> <p>B.</p> <pre>public class TStart extends Runnable{ public static void main(String argv[]){ TStart ts = new TStart(); ts.start(); } public void run(){ System.out.println("Thread starting"); } }</pre> <p>C.</p> <pre>public class TStart extends Thread{ public static void main(String argv[]){ TStart ts = new TStart(); ts.start(); } public void start(){ System.out.println("Thread starting"); } }</pre> <p>D.</p> <pre>public class TStart extends Thread{ public static void main(String argv[]){ TStart ts = new TStart(); ts.run(); } public void run(){</pre>	<p>仅选项 1 是一个有效的方式开始一个新的线程执行。选项 2 的代码继承 Runnable 接口 但没意义,因为 Runnable 是接口不是类, 接口使用 implements 关键字。选项 3 的代码直接地调用起动方法。 如果您运行这个代码您将发现文本输出, 但由于直接调用方法, 并不是因为一个新的线程在运行。 选项 4 也一样, 直接地调用运行线程仅是另一个方法, 并且象其他的一样执行。</p>

	<pre> System.out.println("Thread starting"); } } </pre>	
4	<p>运行以下程序,结果将是:</p> <pre> public class TwoThread extends Thread { public void run() { for(int i=0;i<10;i++) { System.out.print(i); } } public static void main(String[] args) { TwoThread tt=new TwoThread(); tt.run(); for(int i=0;i<10;i++) { System.out.print(i); } } } </pre> <p>A. 01234567890123456789 B. 00112233445566778899 C. 两个 0~9 的序列间插,运行结果可能不同</p>	<p>尽管 run()函数是线程代码执行的起点但要启动一个线程只能调用 start()函数</p>
5	<p>实现了 Runnable 接口的对象, 就是一个线程。()</p> <p>A. 正确 B. 错误</p>	<p>实现 Runnable 接口不等于生成了线程</p>

6	<p>有以下代码，说法错误的是：()</p> <pre>Thread t = new Thread() { public void run() { while(true) { } } }; t.start(); t = null;</pre> <p>A. 这段代码产生了一个线程。 B. 当变量 t 被赋值为 null 后，上述代码产生的线程对象符合垃圾回收的条件。 C. 这段代码生成的线程将消耗极多的资源。 D. 当这段代码执行后，即使 main()函数返回进程也不会终止、除非在别的线程调用 System.exit()方法。</p>	<p>正在运行的线程是由虚拟机控制的，如果线程没有运行结束是不会被释放的</p>
7	<p>当一个 Java 对象的一个 synchronized 方法被调用的时候，该方法就不可能被其他线程调用，但是该 Java 对象的其他 synchronized 方法则可以被调用。()</p> <p>A. 正确 B. 错误</p>	<p>同步基本概念</p>
8	<p>有以下代码，标识为 A/B/C/D 的四行中获得锁的对象与其他的是不同的？(C)</p> <pre>public class Foo { private static synchronized void synchronizedStatic(){} private synchronized void synchronizedInstance(){} public static void main(String[] args) { Foo foo = new Foo(); synchronized(foo.getClass()){} // (A) foo.synchronizedStatic(); // (B) foo.synchronizedInstance(); // (C) synchronized(Foo.class){} // (D) } }</pre>	<p>对象锁和类锁的基本概念 同步锁对类的 class 对象的同步还是对对象的 this 引用进行同步</p>

9	<p>线程同步是为了保证在多线程环境下共享数据的安全,要尽可能多的使用同步机制。()</p> <p>A. 正确</p> <p>B. 错误</p>	同步是有代价的,还要考虑性能及死锁
---	---	-------------------

第9章 util 库

编号	试 题	知识点 (考察点) 说明
1	<p>下面说法不正确的是:</p> <p>A. Collection 是个 java.util 下的接口</p> <p>B. Collections 是 Collection 的子类</p> <p>C. Set、List 继承于 Collection</p> <p>D. Collections 不能被继承</p>	考察 Collections 与 Collection 的区别及两者的基本用法
2	<p>下面说法正确的是:</p> <p>A. Set、List 都是继承于 Collections 的类</p> <p>B. Collections 并非继承于 Collection</p> <p>C. Collections 是个 java.util 下的接口</p> <p>D. Collections 可以被实例化但只能有惟一的实例</p>	考察 Collections 与 Collection 的区别及两者的基本用法
3	<p>以下程序运行结果是:</p> <pre>public class test { public static void main(String args[]) { String[] t1 = {"F", "A", "D", "B", "C", "E"}; Arrays.sort(t1); System.out.println(Arrays.asList(t1)); } }</pre> <p>A. [A, B, C, D, E, F]</p> <p>B. [F, A, D, B, C, E]</p> <p>C. [F, E, D, C, B, A]</p> <p>D. 产生数组越界异常</p>	考察 Arrays 的基本用法, 如常用的 sort 方法和 asList 方法的使用

4	<p>以下程序运行结果可能是:</p> <pre>public class test { public static void main(String args[]) { String[] t1 = {"F", "A", "D", "B", "C", "E"}; Collections.sort(Arrays.asList(t1)); System.out.println(Arrays.asList(t1)); } }</pre> <p>A. [Ljava.lang.String;@eee36c B. [F, A, D, B, C, E] C. [A, B, C, D, E, F] D. 产生数组越界异常</p>	考察 Collections 的基本用法, 如常用的 sort 方法和 asList 方法的使用
5	<p>下面描述错误的是:</p> <p>A. Hashtable 不允许 key 为 null 但允许 vaule 为 null, 而 HashMap 允许将 null 作为一个 entry 的 key 或者 value B. Hashtable 是线程安全的, 也就是说同步的, HashMap 线程不安全的, 不是同步的 C. HashMap 和 Hashtable 都实现了 Map 接口 D. Hashtable 是继承于一个陈旧的类, 多线程时可以对 HashMap 进行同步, 因此在很多场合可以用 HashMap 代替 Hashtable</p>	HashMap 和 Hashtable 的区别和基本概念
6	<p>下面描述正确的是:</p> <p>A. Hashtable 不允许 key 为 null 但允许 vaule 为 null, 而 HashMap 允许将 null 作为一个 entry 的 key 或者 value B. Hashtable 是线程安全的, 也就是说同步的, HashMap 线程不安全的, 不是同步的 C. HashMap 继承于 Hashtable D. Hashtable 要求 key 必须惟一, 但是 HashMap 允许重复的 key 存在</p>	HashMap 和 Hashtable 的区别和基本概念
7	<p>下面说法错误的是:</p> <p>A. ArrayList 在其末尾增删元素与其他位置的增删所花费时间一样, 但 Vector 末尾增删要小于其他位置增删花费的时间。 B. 随着数据的插入, 当内部数组空间不够了 ArrayList 和 Vector 都要扩展它的大小, Vector 默认增加一倍的容量, 而 ArrayList 却是增加 0.5 倍的容量。 C. 对效率优先的代码, 建议使用一个普通的原始的数组来代替 Vector 和 ArrayList。 D. Vector 是同步的, 而 ArrayList 是非同步的。</p>	Vector 和 ArrayList 的区别和基本概念

8	<p>下面程序运行结果描述正确的是:</p> <pre> public class test { public static void main(String[] args) { HashMap hashMap = new HashMap(); hashMap.put("a",null); hashMap.put("a","B"); hashMap.put("a","C"); List result=new ArrayList(hashMap.values()); System.out.println(result); } } </pre> <p>A. [null, B, C] B. [B, C] C. [C] D. 运行出现异常</p>	考察实现 Map 接口的子类存取元素的基本操作
9	<p>下面描述错误的是:</p> <p>A. Map 中 key 和 value 都不允许重复。 B. 当先后添加两个相同的 key 的 key-value 对时, 先添加的 value 将被后添加的 vaule 替代 C. 查看 Map 是否存在指定的“keyX”时可以用方法 containsKey(keyX)来判断 D. 可以通过迭代器遍历 Map 中的 key 和 value</p>	考察 Map 的基本概念
10	<p>下面说法不正确的是:</p> <p>A. 查找一个指定位置的元素, vector 和 arraylist 花费时间要小于 LinkedList B. LinkedList 移动一个指定位置的元素所花费的时间要小于 arraylist C. ArrayList、Vector 和 LinkedList 都实现了 List 接口 D. LinkedList 是线程安全的</p>	考察 ArrayList、Vector 和 LinkedList 之间的却别和基本用法
11	<p>下面说法错误的是:</p> <p>A. Set 拒绝持有重复的元素 B. 不能将一个或多个 null 添加到 HashSet 中 C. 不能通过任何索引的方法来操作 Set 元素 D. HashSet 插入和查找性能优于 TreeSet, 但 TreeSet 能够维护其内元素的排序状态</p>	考察 Set 接口和实现该接口的常用类的特性

12	<p>下面程序运行结果描述正确的是:</p> <pre> public class test { public static void main(String[] args) { List list=new LinkedList(); list.add("a"); list.add("a"); list.add("b"); list.add("b"); list.add(null); list.add(null); HashSet hashSet=new HashSet(); hashSet.addAll(list); System.out.print(list.size()); System.out.print(hashSet.size()); } } </pre> <p>A. 66 B. 33 C. 63 D. 42</p>	考察实现 Set 接口的常用类的特性及基本用法
----	---	-------------------------

第 10 章 Java 开发环境

编号	试 题	知识点 (考察点) 说明
1	<p>下面说法错误的是</p> <p>A. Java 程序在加载某类的时候按照当前包、当前目录中包、classpath 中包的顺序查找。</p> <p>B. 在输出换行动作时, 建议使用 System.getProperty("line.separator")换行, 这可以做到平台无关。</p> <p>C. 在拼接文件名时, 建议使用 System.getProperty("file.separator"), 这可以做到平台无关。</p>	类的加载顺序; 换行; 文件路径


2	<p>文件夹 temp 的目录结构如下</p> <pre> myPackage.jar └── hello Hello.class Hello.java </pre> <p>Hello.java 引用了 myPackage.jar 中内容。在 windows 操作系统中执行,以下可以运行 Hello 程序的命令是()</p> <p>A. java hello/Hello.class</p> <p>B. java hello.Hello</p> <p>C. java -cp myPackage.jar hello.Hello</p> <p>D. java -cp myPackage.jar:. hello.Hello</p>	类的加载顺序
3	<p>如果两个 Jar 包 p1.jar 和 p2.jar 都有类 com.Tool, 但是实现不同。p1.jar 中 com.Tool 有方法</p> <pre> public static void out(String msg) { System.out.println("p1:" + msg); } </pre> <p>p2.jar 中 com.Tool 有方法</p> <pre> public static void out(String msg) { System.out.println("p2:" + msg); } </pre> <p>类 Main 使用 jar 包 p1.jar 编译通过, 其代码如下</p> <pre> public class Main { public static void main(String[] args) { com.Tool.out("Hello"); } } </pre> <p>在 windows 操作系统中执行命令行 java -cp p2.jar:p1.jar:. Main 会输出什么</p> <p>A. 因为 Main 是通过 p1.jar 编译的, 所以输出为 p1:Hello</p> <p>B. 因为-cp 参数种 p2.jar 放在了前面, 所以先找到 p2 中内容, 因此输出 p2:Hello</p> <p>C. -cp 参数中有重复的内容, 系统不能确定使用哪一个, 报错</p> <p>D. 不能确定输出内容, 系统随机选择</p>	类的加载顺序


4	<p>以下说法正确的是()</p> <p>A. 在输出换行动作时, 建议使用 <code>System.getProperty("line.separator")</code>换行, 这可以做到平台无关。</p> <p>B. 在输出换行时最好直接使用"<code>\n</code>", 调用 <code>System.getProperty("line.separator")</code>会影响效率。</p> <p>C. 输出换行时应该使用"<code>\r\n</code>", "<code>\n</code>"只适用于 Linux 系统。</p> <p>D. 全不对</p>	换行
5	<p>假设有一个 <code>pr.properties</code> 文件, 放在 <code>/client/properties</code> 目录下, 在 Solaris 平台下执行下面代码段, 会输出什么?</p> <pre>public static boolean isExist(String path) { File file = new File(path); return file.exists(); } public static void main(String[] args) { String path1 = "/client/properties/pr.properties"; String path2 = "\\client\\properties\\pr.properties"; System.out.println(isExist(path1)+" "+isExist(path2)); }</pre> <p>A. <code>true>true</code></p> <p>B. <code>false>true</code></p> <p>C. <code>true>false</code></p> <p>D. <code>false>false</code></p>	文件路径
6	<p>下面关于文件分割符的说法错误的是</p> <p>A. 各种系统不区分 <code>\</code>和 <code>/</code></p> <p>B. Windows 默认使用 <code>\</code>做为文件分割符</p> <p>C. unix 只能使用 <code>/</code>做为文件分割符</p> <p>D. 使用在需要使用文件分割符的地方使用 <code>System.getProperty("file.separator")</code> 可以保证应用程序的跨平台特性</p>	文件路径
7	<p>某资源文件中有如下内容</p> <pre>#Key1=Value1 Key2 = Value2 Key3 = Value3 Key4 = Super Value4</pre> <p>如下说法正确的是</p> <p>A. 可以正常读出 Key1 为"<code>Value1</code>"</p> <p>B. 可以正常读出 Key2 为"<code>Value2</code>"</p> <p>C. 可以正常读出 Key3 为" <code>Value3</code>"</p>	本地化和国际化

	D. 可以正常读出 Key4 为"SuperValue4"	
--	-------------------------------	--

第 11 章 Java 的 GUI 库

编号	试 题	知识点（考察点）说明
1	javax.swing.JButton 的点击事件使用哪一个监听器接口？（） A. ActionListener actionPerformed(ActionEvent e) B. AdjustmentListener adjustmentValueChanged(AdjustmentEvent e) C. ChangeListener stateChanged(ChangeEvent e) D. ComponentListener componentShown(ComponentEvent e)	事件（监听）
2	不可以将下面哪些监听器接口添加到 JTextArea 对象中？（） A. ComponentListener B. ActionListener C. MouseMotionListener D. MouseListener	事件（监听）
3	下面哪些监听器接口不可以添加到 JTextField 对象中？（） A. ActionListener B. FocusListener C. MouseMotionListener D. WindowListener	事件（监听）
4	下面可以监视 JTextField 内容改变的监听器接口是（） A. ActionListener B. AdjustmentListener C. ChangeListener D. DocumentListener	事件（监听）

5	<p>关于 javax.swing.event.ListSelectionListener 事件参数 javax.swing.event.ListSelectionEvent 的 getValueIsAdjusting()方法中错误说法是()</p> <p>A. 使用鼠标选择内容时, 如果只选择一条记录, 则选择改变事件只被触发一次, 且 getValueIsAdjusting()等于 false</p> <p>B. 鼠标在按下之后触发的任何与选择有关的动作, 都会触发改变事件, 且 e.getValueIsAdjusting()都等于 true。</p> <p>C. 鼠标在选择多条记录后, 松开左键会触发一次改变事件, 且 e.getValueIsAdjusting()等于 false.</p> <p>D. 使用键盘改变选择内容时, 选择改变事件只被触发一次, 且 javax.swing.event.ListSelectionListener 等于 false</p>	事件 (监听)
6	<p>下面哪种类型的 model 承载了 JTextField 的实际内容? ()</p> <p>A. TableModel</p> <p>B. ListModel</p> <p>C. Document</p> <p>D. TreeModel</p>	数据 (模型)
7	<p>哪些 swing 组件使用 ListSelectionModel? ()</p> <p>A. JList and JComboBox</p> <p>B. JPopupMenu and JTable</p> <p>C. JTable and JComboBox</p> <p>D. JList and JTable</p>	数据 (模型)
8	<p>在下列布局管理器中, 哪个布局管理器使用东西南北中方式放置控件? ()</p> <p>A. BorderLayout</p> <p>B. CardLayout</p> <p>C. FlowLayout</p> <p>D. GridBagLayout</p>	BorderLayout
9	<p>在下列布局管理器中, 哪种布局管理器从左到右、从上到下排列组件? ()</p> <p>A. BorderLayout</p> <p>B. CardLayout</p> <p>C. FlowLayout</p> <p>D. GridBagLayout</p>	FlowLayout
10	<p>容器中要放置一行组件, 这些组件排列方式是在一行中并且大小相同而且充满整个容器空间 (中间没有间隔), 如下图所示, 请问使用哪个 Layout 管理器比较合适? ()</p> <p>A. BorderLayout</p> <p>B. CardLayout</p> <p>C. FlowLayout</p> 	GridLayout

	D. GridLayout	
11	<p>在以下布局管理器中，哪种布局管理器像电子表格一样，可以实现用行列来排列组件？ ()</p> <p>A. BorderLayout</p> <p>B. CardLayout</p> <p>C. FlowLayout</p> <p>D. GridBagLayout</p>	GirdBagLayout
12	<p>如下窗口采用 GridBagLayout 布局器，想达到输入框控件(nameTextField)随着窗口缩放而变化长度，标签姓名控件(nameLabel)宽度不变化，以下哪个方法可以实现？</p>  <p>A. 设置 nameLabel 所在 GridBagConstraints 的 weightx 属性为 0; 设置 nameTextField 所在 GridBagConstraints 的 weightx 属性为 1。</p> <p>B. 设置 nameLabel 所在 GridBagConstraints 的 weightx 属性为 1; 设置 nameTextField 所在 GridBagConstraints 的 weightx 属性为 2;</p> <p>C. 设置 nameLabel 所在 GridBagConstraints 的 gridwidth 属性为 1; 设置 nameTextField 所在 GridBagConstraints 的 gridwidth 属性为 2;</p> <p>D. 设置 nameLabel 所在 GridBagConstraints 的 fill 属性为 NONE; 设置 nameTextField 所在 GridBagConstraints 的 fill 属性为 HORIZONTAL;</p>	GirdBagLayout

13	<p>关于 GridBagLayout 说法错误的是</p> <p>A. anchor 用于控制控件在自己所在格子中出现的位置</p> <p>B. fill 说明控件在格子中填充的方式，设置了填充时，控件的 size 就会失效</p> <p>C. gridheight 用于控制控件所在格子的高度，单位是像素</p> <p>D. gridx 用于设置控件所在格子的水平位置</p>	GridBagLayout
14	<p>关于 GridBagLayout 说法错误的是</p> <p>A. insets 属性用于控制控件距离四边距离，其所在格子四边的距离，单位像素</p> <p>B. weightx 属性设置控件所在格子的权重，此值越大，分得的空间越多,等于 0 时就隐藏了</p> <p>C. gridheight 用于控制控件所在格子在竖直方向上合并多少行</p> <p>D. fill 设置为 none 时，控件的大小使用 preferredSize</p>	GridBagLayout
15	<p>采用以下措施,可以很好的抑制 Swing 应用程序灰屏的是 ()</p> <p>A. 不要在事件派发线程中执行耗时操作</p> <p>B. 将耗时操作放到 invokeLater()方法中</p> <p>C. 在事件派发线程中正确地使用 invokeLater()和 invokeAndWait()方法</p>	invokeLater() 和 invokeAndWait()函数的作用
16	<p>造成 Swing 应用程序灰屏的根本原因是()</p> <p>A. 在非事件派发线程中没有使用 invokeLater()和 invokeAndWait()方法</p> <p>B. 事件派发线程被阻塞</p> <p>C. 主线程被阻塞</p>	灰屏现象
17	<p>GUI 的操作应该在什么线程中才能保证是线程安全的?()</p> <p>A. 普通线程</p> <p>B. 主线程</p> <p>C. 事件派发线程</p> <p>D. 任何情况都是安全的</p>	Swing 中最安全的 GUI 操作方式就是通过事件派发线程，保证不会发生界面显示问题。

18	<p>不考虑程序的完整性,以下三个按钮事件,哪个处理的更安全?</p> <p>A.</p> <pre>btn1.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent ev) { resultLabel.setText("Working . . ."); setEnabled(false); Thread worker = new Thread() { public void run() { 这里执行耗时操作... SwingUtilities.invokeLater(new Runnable() { public void run() { resultLabel.setText("Ready"); setEnabled(true); } }); } }); worker.start(); } });</pre> <p>B.</p> <pre>btn2.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent ev) { resultLabel.setText("Working . . ."); setEnabled(false); 这里执行耗时操作... resultLabel.setText("Ready"); setEnabled(true); } });</pre>	<p>invokeLater() 和 invokeAndWait()函数 的作用</p>
----	---	--

	<pre> C. btn3.addActionListener(new ActionListener() { public void actionPerformed(ActionEvent ev) { resultLabel.setText("Working . . . "); setEnabled(false); SwingUtilities.invokeLater(new Runnable() { public void run() { 这里执行耗时操作... resultLabel.setText("Ready"); setEnabled(true); } }); } }); </pre>	
--	---	--

第 12 章 Web

编号	试 题	知识点（考察点）说明
1	<p>如下那种方法不可以用来跟踪客户的状态（）</p> <p>A.session B.url rewriting C.cookie D.response E.hidden field</p>	web client 状态跟踪
2	<p>由于 Http 协议是无状态，采用如下的哪个方法保存客户状态的数据，没有大小限制，而且性能表现最好（）</p> <p>A. session B. cookie C. url rewriting D. hiddenfiled</p>	web client 状态跟踪

3	<p>关于 Servlet 中 doGet()和 doPost()的说法不正确的是()</p> <p>A. doGet()和 doPost()都可以用来处理业务逻辑</p> <p>B. doGet()对于提交的数据有限制, doPost()则没有限制</p> <p>C. doPost()比 doGet()提交数据更安全, 因为参数封装 在请求体中</p> <p>D. 表单提交只能使用 doPost()</p>	Get 和 Post 的区别
4	<p>如下的代码片断</p> <pre><html><body>POST</body></html></pre> <p>点击上述页面中的超链接, helloservlet 中的那个方法会被调用 ()</p> <p>A. doGet() B. doPost() C. doForm() D. serviceGet()</p>	Get 和 Post 的区别
5	<p>Web 容器初始化一个 Servlet 的时候, 会先调用如下的哪个方法 ()</p> <p>A. init() B. service() C. destroy() D. 如上答案都不对</p>	servlet 生命周期的几个方法
6	<p>如果希望 Servlet 退出的时候能够执行相关的资源清除, 可以在 Servlet 中的那个方法中处理 ()</p> <p>A. init() B. service() C. destroy() D. close()</p>	servlet 生命周期的几个方法
7	<p>如下哪些选项不属于 JSP 页面中的元素()</p> <p>A. 标签</p> <p>B. 脚本</p> <p>C. 隐含对象</p> <p>D. 插件</p>	jsp 页面中的元素
8	<p>在 JSP 页面中可以包含如下的元素:jsp 指令、Jsp 声明, 隐藏对象, 自定义的标签等()</p> <p>A. 正确 B. 错误</p>	jsp 页面中的元素
9	<p>在 JSP 页面中引用 javabean 的正确形式是 ()</p> <p>A. <jsp:useBean id="name" class="package.class" scope="request"></p> <p>B. <file:useBean id="name" class="package.class" scope="page" ></p> <p>C. <page:useBean id="name" class="package.class" scope="session" ></p> <p>D. <jsp:useBean id="name" class="package.class" scope="response" ></p>	JSP 页面如何引用 JavaBean,设置 bean 的范围
10	<p>Serlvet 中的 forward 是在服务端执行, 不会产生新的请求()</p> <p>A. 正确 B. 错误</p>	forward 和 redirect 的区别

11	Serlvet 中的 redirect 是在服务端执行, 不会产生新的请求() A. 正确 B. 错误	forward 和 redirect 的区别
12	Servlet 获取请求的参数值, 如下那个方法是正确的 () A. getAttribute() B. getParamter() C. getAttributeValue() D. getHeader()	request 对象常用的方法
13	在 Servlet 中处理请求时取得消息头信息, 可以使用如下的那个方法 () A. getAttribute() B. getParamter() C. getParameterValues() D. getHeader()	request 对象常用的方法
14	下面关于静态包含和动态包含的哪个说法是错误的 () A. 静态包含是在编译期完成, 编译后内容不可变, 执行效率高 B. 动态包含的内容是动态变化的, 在执行期才确定 C. 如果包含的页面不经常变动, 采用静态包含较好 D. <%@ include file="included.jsp" %> 是动态包含	静态包含和动态包含的区别
15	以下哪一种描述表示 JSP 页面中的动态包含 () A. <jsp:include page="included.jsp" flush="true"/> B. <%@ include file="included.jsp" %> C. <%page include file="included.jsp" flush="true"%> D. <jsp:include file="included.jsp"> <jsp:include/>	静态包含和动态包含的区别
16	下面关于过滤器的说法那个是不正确的() A. 过滤器可以拦截 request 和 response B. javax.servlet.Filter 是一个抽象类 C. 过滤器可以串联使用 D. 需要在 web.xml 中设置才可以启用过滤器	过滤器
17	下面关于过滤器的说法那个是不正确的() A. 通过过滤器中, 我们可以实现统一的页面编码设置、访问日志记录、页面访问权限验证等 B. javax.servlet.Filter 是一个接口 C. 过滤器可以串联使用 D. 以上说法都不对	过滤器