须知与说明：

1、 该文档资料，均来源于网络，具有一定的参照价值；

2、 该文档资料，便于学员把握复习要点、掌握提醒思路等；

3、 该文档资料，并不代表真实题库及试题内容；

4、 内部资料，请勿外传；

1. 如果你用public 关键字在一个文件中定义了超过一个非内部类，编译器将会报错。

2. protected修饰符的访问权限:能修饰属性、方法和构造函数。同一个包内部可见。与基类不在同一个包中的子类，只能访问自身从基类继承而来的受保护成员，而不能访问基类实例本身的受保护成员。保护成员对除了子类外的所有类实际上都是包级访问或默认访问。对于包外子类，只能通过继承访问收保护成员。

3. 通过参数的顺序和类型来区分两个重载的方法。返回类型对区分方法没有帮助。你可以在一个子类中重载一个方法，所需要的就是新方法有不同的参数顺序和类型。参数的名字或者返回类型都不作考虑。，重写后的方法必须跟基类中被取代的原始方法有完全相同的签名。这就包括了返回值。static 方法不能被重写。

4. default 语句不是必须在case 语句的结尾处出现。switch 语句的参数必须是一个byte，char，short 或int 类型的变量。

5. 。if 表达式的要求是必须返回一个boolean 类型，因为(b=false)返回一个boolean 类型，所以被接受（如果无用处）.

6. for中定义的变量对for 循环外部来说永远是不可见的。

7. 在子类中一个重写的方法可能只抛出父类中声明过的异常或者异常的子类，或者不抛出异常。这只适用于方法重写而不适用于方法重载。子类重写方法的修饰符不能低于父类方法的修饰符。

8. 一个类的构造方法无论是什么修饰符去修饰，它main方法中都可以去实例化这个类。但其他类要实例化这个类就和构造方法的修饰符有关了。一下代码可以通过编译:

```
import java.io.*;
class Base{
public static void amethod()throws FileNotFoundException{}
}
46
public class ExcepDemo extends Base{
public static void main(String argv[]){
  ExcepDemo e = new ExcepDemo();
}
public static void amethod(int i)throws IOException{}
private ExcepDemo(){
try{
DataInputStream din = new DataInputStream(System.in);
System.out.println("Pausing");
din.readChar();
System.out.println("Continuing");
this.amethod();
```

}catch(IOException ioe) {}
}
}

9. System.gc()来建议垃圾回收器收集垃圾，但是这并不能保证执行。当代码已经无法再访问对象的时候，这个对象就成为了可垃圾回收的。有两种情况下会出现对象无法再被访问，第一，对象的引用设置为null；第二，指向这个对象的引用指向了
其他的对象。

10. 使用引入子句对性能没有影响。这类似于在 DOS（或Unix）环境中设定一个路径声明。这只是简单的为类设定有效性或路径，并不是直接将代码引入程序中。仅仅在程序中
实际地使用类才会影响性能。

11. 定义在方法中的类只能看到来自嵌套方法中的final 域。但是它可以看到嵌套类中包括私有域在内的域。

12. 类构造函数的执行顺序：由此不难看出java类初始化时构造函数调用顺序：

（1）初始化对象的存储空间为零或null值；
（2）按顺序分别调用父类成员变量和实例成员变量的初始化表达式；
（3）调用父类构造函数；（如果使用super()方法指定具体的某个父类构造函数则使用指定的那个父类构造函数）
（4）按顺序分别调用类成员变量和实例成员变量的初始化表达式；
（5）调用类本身构造函数。

13. java中的关键字：

abstract	boolean	break	byte	case	catch
char	class	const *	continue	default	do
double	else	extends	final	finally	float
for	goto *	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while

14. ，类级别的变量总是会被赋予一个缺省值，而一个成员变量（包含在方法中）将不会被赋予任何缺省值。

15. 任何基本类型的数组元素的值将总是被初始化为缺省值，无论数组是否被定义。无论数组定义为类级别还是方法级别，元素值都会被设定为缺省值.

16. java三种进制的表示：

Decimal	18
Octal	022 (Zero not letter O)
Hexadecimal	0x12

三种方式以及字母是非大小写敏感的要更容易一点。

17. 一个具有小数部分的数据的缺省类型是 double 而不是float。

18. char 是Java 中唯一的未赋值的原始数据类型，它是16 位长的。

19. 对于基本类型的包装类可以直接赋值。

20. 只有+运算符可以对字符串进行重载，如果你对字符串使用除号和减号（/ -），你会得到一个错误。

21. 3 >>> 32的值是3，也就是说3 被移动了0 位。

22. 参数传递：基本类型是值传递。引用类型是地址传递，可以改变其内容。

23. 运算优先级：单操作数运算符>算数运算符>移位运算符>比较运算符>按位运算符符>逻辑运算符>条件运算符>赋值运算符.

24. 二进制负数转成整数规则：-1(减1)→取反(~)→添上符号(-)

25. 负整数转成二进制规则:取模→取反(~)→+1(加1)

26. 带符号移位运算:

左移：相当于*2
右移：相当于/2

27. 逻辑运算符（&&、||）如果你理解AND，你就会理解Java 的方法，如果第一个操作数为假，第二个操作数的值就没有作用了，所有的结构都为假。对于逻辑OR 也是，如果第一个操作数为真，所有的计算结果将为真，因为只要一个操作数为真最后的结果就为真

28. 按位运算符符可以对boolean进行操作.

29. 按位运算优先级顺序：&>^>|.方法：只有当两个变量相同位置上的位都是1时，&运算符才将结果位设置为1；只要有一个变量的位是1，|运算符就将结果位设置为1；仅当只有其中一个变量的位是1时,^运算符才将结果位设置为1.

30. final常量在使用前必须被初始化，否则会产生编译错误。

31. static关键字不能修饰顶层类。方法中的局部变量不能使用static来修饰。

32. transient关键字只能用于修饰变量，而不能修饰类和方法。注意：本地变量是不能被transient关键字修饰的。

33. synchronized关键字不能修饰构造函数。

34. 单操作数运算符在进行算数时的基本数据类型转换规则如下：

1）.当运算符为取正运算符(+)、取负运算符(-)、或按位取反运算符(~)时，如果操作数为字节型(byte)、短整型(short)、或字符型(char)、则先被转换为整型(int)，再参与算术运算。

2）.当运算符为自动递增运算符(++)或自动递减运算符(--)时，如果如果操作数为字节型(byte)、短整型(short)、或字符型(char)、则不用先被转换成整型(int),而是直接参与算数运算，且运算结果类型也不变。

3）.如果操作数为整型(int)或长整型(long),则无论运算符为何种单操作数运算，均不发生类型转换，直接参与算术运算，且运算结果类型不变。

双操作数运算符在进行算数时的基本数据类型转换规则如下：

1）.如果操作数之一为双精度型(double)，则另一个操作数先被转换为双精度型(dounle)，再参与算术运算。

2）.如果两个操作数均不为双精度(double)，当操作数之一为单精度型(float)，则另一个操作数先转换为单精度(float)，再参与算术运算。

3）. 如果两个操作数均不为双精度(double)或者单精度型(float)，当操作数之一为单长整型(long)，则另一个操作数先转换为长整型(long)，再参与算术运算。

4）. 如果两个操作数均不为双精度(double)、单精度型(float)或者长整型(long)，则两个个操作数先转换为整型(int)，再参与算术运算。

5）.如果采用+=、*=等缩略形式的运算符，系统会自动强制将运算结果转换成目标变量的类型。

35. 数组实现了Cloneable接口，继承了Object类。

36. 只有执行System.exit()方法或出现error错误时，finally语句块才不会获得执行而退出程序。

37. 在使用try,catch块时，如果try中没有响应的异常产生，则catch中不能有这种异常。否则编译错误。但exception、throwable可以出现。运行时异常也可以，如NullPointException.

38. 注意到在用实现 Runnble 接口的方法来创建线程时，必须要求创建一个Thread 对象

的实例，并且必须要在创建的时候，把实现该Runnable 接口的对象作为构造方法的参数传递进去。

39. 关键字 synchronized 可以用在标记一段声明或者锁定一段代码，请注意的是这里的锁是基于对象而不是基于方法的。

40. wait 和 notify 应该放在 Synchronized 关键字标记的代码中以保证当前的代码在监视器的监控之中。

41. 线程调度不是独立的，不能依靠虚拟机让它用同一种方式运行。

42. **ceil方法:** 这个方法返回的是比被操作数大的最小double 值。比如下面这个例子：
ceil(1.1)
它将返回2.0
如果你换成
ceil(-1.1)
它将返回 -1.0；

43. **floor方法:** 返回最大的（最接近正无穷大）double 值，该值小于或等于参数，并且等于某个整数。
如果觉得这表达的不够清楚，那么我们可以看一下，下面那一小段代码和它的输出情况：

```
public class MyMat{
public static void main(String[] argv){
System.out.println(Math.floor(-99.1));
System.out.println(Math.floor(-99));
System.out.println(Math.floor(99));
System.out.println(Math.floor(-.01));
System.out.println(Math.floor(0.1));
}
}
```

它的输出是：
-100.0-99.0
99.0
-1.0
0.0

44. 不像一些随机数系统，Java 似乎并不支持提供种子数来增加随机性。

45. **round方法:** 返回最接近参数的一个整型数。如果小数部分大于0.5 则返回下一个相对最小整数，如
果小数部分小于等于0.5 则返回上一个相对最大整数。如下例所示：
2.0 <=x < 2.5. then Math.round(x)==2.0
2.5 <=x < 3.0 the Math.round(x)==3.0

以下是一些例子和它们的输出：

```
System.out.println(Math.round(1.01));
System.out.println(Math.round(-2.1));
System.out.println(Math.round(20));
```

1
-2
20

46. **sin cos tan方法:** 这三个方便快捷的方法都只需要一个 double 型的参数，它们的功能和其他语言里面的方法功能是一样的

47. **Set**任何一个传递给 add 方法的对象必须实现equals 方法，这样保证与已存数据进行对比。如果已经存在该数据，那么调用add 方法不会对该set 起任何影响并且返回false。

48. 接口可以是默认的访问修饰符。

49. 抽象类的第一个具体子类必须实现超类的所有抽象方法。一个方法永远不能同事标识为abstract和final，或者同时标识为abstract和private，也不能与static修饰符组合使用。

50. static不能修饰类。

51. 从java5起，只要新的返回类型是被重写的(超类)方法所声明的返回类型的子类型，就允许更改重写方法中的返回类型。

52. 只能将静态变量和方法作为调用super()或this()的一部分进行访问。例如 super(Animal.NAME);不能用非静态属性.

53. 一个类的构造方法是私有的时候，这个类的main方法是可以访问的.在main方法中可以直接new一个类的实例.

54. 以下赋值是非法的：

```
Int x=25,36;
Char e=-29;
Char e=70000;
```

以下赋值是合法的：

```
Int x,y=1;
Char e=(char)-29;
```

Char e=(char)70000;

55. 以下声明二维数组是合法的。Int [][] myArray=new int[2][];JVM只需要知道赋予变量myArray的对象大小。

56. 使用匿名数组创建语法时，不能指定数组的大小。new int[]{1,2}.

57. 所有数组均是对象。Int数组不能引用其他类型的数组。以下代码是错误的:
char[] zz=new int[2];

57. 实例初始化块代码的运行时间恰好发生在调用构造函数中super()之后。换句话说，在所有超类构造函数运行之后。如果有多个初始化块，则运行顺序是自顶向下。静态代码块只会在类初始化的时候执行一次。

58. 除了Character之外，所有的包装类都提供2个构造函数：一个是基本类型；另一个是String表示的变元。

59. 为了节省内存，对于下列包装器对象的两个实例，当它们的基本值相同时，它们总是==关系：

Boolean
Byte
从\u0000到\u007f的字符(7f是十进制的127)。
-128~127的Short和Integer

60. 对于方法的重载需要记住以下规则：

加宽优先于装箱执行
加宽优先与var-arg(变长参数)执行
装箱优先于var-arg(变长参数)执行
不能从一个包装类加宽到另一个包装类
不能先加宽，后装箱
可以先装箱，后加宽
顺序是：
1.参数是基本类型：先匹配自身>加宽>包装类
2.参数是包装类型：先匹配自身>拆箱基本类型>拆箱加宽基本类型

61. 在final变量上不能使用递增或递减运算符.

62. case常量必须是编译时常量.一下代码是错误的:

```
Byte g=2;
Switch(g){
    Case 128:
}
```

128对于byte来说太大了.

多个case标签使用相同的值也是非法的。

63. switch中的default会像任何其他case一样直落运行。

64. 使用不带catch子句和finally子句的try子句是非法的。任何catch子句都必须紧跟在try块之后。任何finally子句都必须紧跟在最后一个catch子句之后(或者如果没有catch子句，则必须紧跟在try块之后)。不能在try、catch块，或者try、finally块之间加入任何代码。
Exception、Error、RuntimeException和Throwable类型的异常都能够使用throw关键字抛出。

65. printf的用法:

字符串的构造规则:
%[arg_index$][flags][width][.precision]conversion char
方括号里的值是可选的。2$表示第二个conversion char.
arg_index:一个整数，其后紧跟一个$,它指定应该在这个位置输出的变元。
flags:
-:左对齐变元
+：让变元包含符号(+或-)
0：用0填变元
，：用于特定地区的分组分隔符(比如“123，456”中的逗号)
(：将负数包含在括号内
width:指定要输出的最少字符数.
precision:精度
conversion：格式化变元的类型
b:布尔型
c：字符型
d:整型
f:浮点型
s:字符串
示例:

```
int i1=-123;
int i2=123456;
System.out.printf(">%1$(7d< \n",i1);
System.out.printf(">%0,7d< \n",i2);
System.out.printf(">%+-7d< \n",i2);
System.out.printf(">%2$b + %1$5d< \n",i1,false);
```

结果:

```
>  (123)<
>123,456<
>+123456<
```

```
>false +  -123<
```

记住：如果转换字符和变元中指定的类型不匹配，就会得到一个运行时异常：
Exception in thread "main" java.util.IllegalFormatConversionException: d != java.lang.Double

66.可排序集合(TreeSet、TreeMap)方法介绍：

lower(e)/lowerKey(key)与floor(e)/floorFey(key)的差别：前者返回比给定元素小的元素，后者返回比给定元素小或者相等的元素。
higher(e)/higherKey(key)与ceiling(e)/ceiling(key)的差别:前者方法返回比给定元素大的元素，后者返回比给定元素大或者相等的元素。
pollFirst()/pollFirstEntry():返回并删除第一项
pollLast()/pollLastEntry():返回并删除最后一项
descendingSet()/descendingMap():以返序返回NavigableSet.

对于副本集合调用pollFirstXxx()几乎总会删除两个集合中的项，但对于原始集合调用pollFirstXxx()，则只会删除原始集合中的项。

67. priorityQueue类：

offer(e): 加载队列。
peek():返回最高优先级的元素但不删除它。
poll():返回最高优先级元素并删除它。

68.自然排序的一些细节：空格排在字符前、大写字符排在小写字符前。

69. TreeSet 集合放入的对象要求是能进行比较排序的。如下代码:

```
Set set =new    TreeSet();
            set.add("2");
            set.add(3);
            set.add("1");
            Iterator it=set.iterator();
            System.out.println(set.size());
            while(it.hasNext()){
                  System.out.println(it.next());
            }
```

向 TreeSet 中放入的元素有 String 和 Integer 两种类型，不能进行比较。在编译时期不会有问题。但会有运行时异常：Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer.

70. Object 类的 hashCode()方法是 native 修饰的。Map 中的键值是不润许重复的,判断是否重复的一个条件是判断 hashCode.不重写 hashCode()方法会导致相同对象含有不同的

hashCode.

71. add()方法是接口 Collection 的方法。所以实现了这个接口的类都有这个方法。类 PriorityQueue 一个基于优先级堆的无界优先级队列。优先级队列的元素按照其自然顺序进行排序，或者根据构造队列时提供的 Comparator 进行排序，具体取决于所使用的构造方法。优先级队列不允许使用 null 元素。依靠自然顺序的优先级队列还不允许插入不可比较的对象（这样做可能导致 ClassCastException）。

72. binarySearch()方法需要注意:

1.必须进行排序。即 Arrays.sort.否则返回的值是不确定的.不会出现编译错误。
2.如果定义了排序的方法，并在 Arrays.sort 方法中使用则在 binarySearch()中也要使用这个参数，否则会返回-1.

73.关键字 this 表示当前正在执行的对象.在内部类中 this 表示内部类，而外部类的表示则是 MyOuter.this。

74. 如下修饰符能够应用于内部类：

1.final
2.abstract
3.public
4.private
5.protected
6.static
7.strictfp

75.在一个静态方法内声明的局部类只能访问该封装类的静态成员。如果位于一个没有"this"的静态方法内，则该方法内的内部类的限制与静态方法相同。换句话说，根本不能访问实例变量。

76.在看到匿名内部类的时候，要格外的注意结束时除了有波形括号外还有一个分号。

77. 如果试图在匿名内部类引用上调用超类没有定义的方法，编译器会发出警告。静态内部类不能访问实例变量和类的非静态方法。

78. 只要线程已经启动过，它就永远不能再次启动。当线程的目标 run()方法结束时，该线程就完成了。Java 规范中根本没有提到线程将按照他们启动的顺序【也就是在每个线程上调用 start()的顺序】开始运行。

79. 能使运行中线程离开运行中状态的 3 种方法：

Sleep():保证使当前线程至少在指定的睡眠期间停止执行(尽管它可能在指定的时间之前被中断)。
Yield():不能保证做太多事情，尽管通常它会使当前的运行中线程移回可运行状态，并使具有相同优先级的线程能够有机会运行。
Join():保证当前线程停止执行，直到它所加入的线程完成为止。然而，如果它加入的线程不是活的，则当前线程不需要停止。

80. import 语句每次只能导入一个包。如果将 jar 文件放入 ext 子目录(.../jar/lib/ext 目录)，则 java 和 javac 可以找到它们并使用它们所包含的类文件。

81. 在使用 java 命令行编写代码时，-classpath 选项必须出现在类名前。-ea 参数表示启用断言。泛型是在 java5 中才引用的。-source 参数后面可以跟版本

82. 使用-classpath 选项查找类的时候，参数从左向右运行。Unix 下使用":"分隔目录参数

83. -D 选项不是一个编译器标志，不能使用 javac 命令。而且与-D 相关的"名称=值"对必须紧跟着它的后面，中间不能有空格。

84. 使用静态导入的正确格式：import static java.lang.Integer.*;或者 import static java.lang.Integer.MAX_VALUE;

85.以下是与 jar 文件结构有关的几条规则:

Jar 命令会自动创建 META-INF 目录。
Jar 命令会自动创建 MANIFEST.MF 文件。
Jar 命令不会将你的任何文件放入 META-INF/目录中。
从以上规则可以看出，在 jar 文件中准确体现了树结构。
Java 和 javac 命令会像普通的目录树一样使用 jar 文件。

86. java 和 javac 都使用相同的基本查找算法：

1.它们都具有同样的查找位置(目录)清单，用于查找类。
2.它们都以同样的顺序查找遍历这个目录清单。
3.一旦发现所查找的类，就不会再查找这个类了。如果查找清单包含具有相同名称的两个或多个文件，则会使用找到的第一个文件。
4.查找的首选位置是包含标准 J2SE 类的目录中。
5.查找的次要位置是在由类路径定义的目录中。
6.类路径应该被看作"类查找路径"，它们是可以在其中找到类的目录清单。
7.有两个地方可以声明类路径。类路径可以声明为操作系统的一个环境变量。只要调用 java 和 javac，就会默认使用作为环境变量声明的路径。类路径作为可以声明 java 或 javac 的命令选项。作为命令行选项声明的类路径，会覆盖作为环境变量声明的类路径，但命令行中的类路径只会在调用期间存在。

87.在使用多线程时，如果使用的对象属性不是 static 时，可能会出现运行时异常。静态属

性不会出现这种状况。
如下代码，运行时会出现 java.lang.NullPointerException

```
import java.util.List;

public class ChicksYack implements Runnable {
    List<String> adds;
    public void run() {
        adds.add("a");

    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        new ChicksYack().go();

    }

    public void go(){
        adds=new ArrayList<String>();
        new Thread(new ChicksYack()).start();
        new Thread(new ChicksYack()).start();
    }

}
```

88. Tread 的 join()方法必须放置在 try/catch 块中。它的作用是：会使主线程暂停并加入到另一个线程的末尾。

89.StringBuffer 天生就是同步的，但多次在一个方法中调用 append()方法时，多线程不能阻止它混和消息。

90.当在对象上调用 wait()方法时，执行该代码的线程会立刻放弃它在对象上的锁。然而，当调用 notify()时，并不意味着此时线程会放弃其锁。如果线程仍然在完成同步代码，则线程在移出同步代码之前不会放弃锁。因此，只调用 notify()并不意味着这时锁将变得可用。

91.主要的线程方法:斜体是静态方法

Object 类	Thread 类	Runnable 类
Wait()	Start()	Run()
Notify()	*Yield*()	
notifyAll()	*Sleep*()	
	Join()	

必须在同步方法内调用 wait()、notify()和 notifyAll()！线程不能调用对象上的等待或通知方法，除非它拥有该对象的锁。

92. 静态同步方法和非静态同步方法将不会彼此阻塞-它们可以同时运行。

93. 如果线程进入了可运行状态，而且它比池中的任何线程以及当前运行中线程具有更高的优先级，则具有较低优先级的运行中线程通常将回撤到运行状态，而具有最高优先级的线程将被选择运行。
另一种没有保证的操作发生在这种情形：当池内的线程具有相同的优先级，或者当前运行中线程与池内的线程具有相同的优先级时。

94. 由于 sleep()到期而醒来，并不意味着它将返回到可运行状态！记住，当线程醒来时，它只会返回到可运行状态。因此，sleep()中指定的时间长度是线程不会运行的最短时间，而不是线程不会运行的实际时间长度。

95.当线程已经被实例化但没有启动【换句话说，还没有在 Thread 实例上调用 start()方法】，就称该线程处于新状态之中。在这个阶段，还不能认为线程是活着的(alive)。一旦调用了 start()方法，线程就被认为是活着的【即使 run()方法可能还没有实际开始执行】。在 run()方法结束时，线程被认为是死的(不再活着)。

95.Thread 类本身也实现了 Runnable【不要忘了，它有一个我们正在重写的 run()方法】。这意味着能将一个 Thread 传递给另一个 Thread 的构造函数：
Thread t=new Thread(new MyThread());
尽管这样做毫无意义，但它是合法的。在这种情况下，真正需要的只不过是 Runnable,完全没有必要创建另一个完整的 Thread。

96.静态方法不能调用非静态内部类。类似静态方法不能调用非静态成员变量。

97.main 方法后面可以有 throws Exception。以下是正确的：

```
public static void main(String[] args) throws Exception{
    System.out.println("ok");

}
```

98.注意就近原则,不会调用其后面的实例：

```
public class A {
    public static void main(String[] args) throws Exception{
        new A().go();

    }

    void go(){
        new B().m();
         class B{
                void m(){
                    System.out.println("inner");
                }
            };
    }
    class B{
        void m(){
            System.out.println("middle");
        }
    }
}
```

最后的输出是:middle.

99.除了匿名内部类这种情况外，不要被任何试图实例化接口的代码所蒙骗。下面的代码是非法的:

Runnable r =new Runable();//can't instantiate interface

而下面的代码是合法的，因为它是在实例化 Runnable(匿名实现类)的实现器:

```
Runnable r =new Runable(){
    Public void run(){}
};
```

如果试图在匿名内部类引用上调用超类定义中没有定义的任何方法，编译器就会发出警告。

100.局部方法内部对象不能使用它所在的方法的局部变量。在一个静态方法内声明的局部类只能访问该封装类的静态成员，因为没有与该封装类相关的任何实例。如果位于一个没有"this"的静态方法内，则该方法内的内部类的限制与静态方法相同。换句话说，根本不能访问实例变量。

101.内部类可以访问外部类的私有成员，因为内部类也是外部类的一个成员。

102.当使用通配符时，List<? Extends Dog> 表示可以访问集合但不能修改它。当使用通配符时，List<?>表示任何泛型类型都可以赋给引用，但只能访问，不能修改。

103.如果 x.equals(y)为 false，则 x.hashCode()==y.hashCode()可以是 true 或 false。如果 x.equals(y)为 true，则 x.hashCode()==y.hashCode()为 true。

104.如果 x 不为 null,则 x.equals(null)为 false.

105.如果看到通配符号(即问号)，它就意味着”许多可能性”；如果没有看到问号,则表示尖括号中的<type>，并且绝对不是其他东西。List<Dog> 就是 List<Dog>,而不是 List<Beagle>、List<Poodle>或 Dog 的任何其他子类。以下是非法的：

```
List<Test_001> xx=new ArrayList<Test_002>();
```

但可以添加子类元素。以下是合法的:

```
List<Test_001> xx=new ArrayList<Test_001>();
Test_002 test_002=new Test_002();
xx.add(test_002);
```

但 List<? extends Dog>可以表示 List<Beagle>、List<Poodle>。
通配符还存在另外一种使用场景，而且仍然可以将其添加到集合，不过是以安全的方式进行的，这就是关键字 super。当使用<? Super ...>语法时，就是在告知编译器你能接受 super 右边的类型或它的任何超类型。

106.数组存在一个运行时异常(ArrayStoreException)，它会阻止将错误的对象类型放入数组中，所以可以成功编译。

107.假设已经通过 tailXxx()或 subXxx()方法创建一个”后备集合”，则通常而言，原始集合与复制的集合具有不同的”第一个”元素。对于考试，要重点记住的是,pollFirstXxx()方法将总是从调用的集合中删除第一项,但在其他集合中,则只会删除具有相同值的那个元素。因此,对于副本集合调用 pollFirstXxx()方法几乎总会删除两个集合中的项，但对于原始集合调用 pollFirstXxx()，则只会删除原始集合中的项。如果试图将一个超出范围的项添加到复制的集合中，则会抛出异常。

108.TreeSet 和 TreeMap 中重要的”后备集合”方法:

方法	描述
headSet(e,b*)	返回一个子集，以元素 e 结尾，不包含 e
headMap(k,b*)	返回一个子映射,以键 k 结尾，不包含 k
tailSet(e,b*)	返回一个子集，从元素 e 开始,包含 e
tailMap(k,b*)	返回一个子映射,以键 k 结尾，包含 k
subSet(s,b*,e,b*)	返回一个子集，从元素 s 开始,在元素 e 之前结束
subMap(s,b*,e,b*)	返回一个子映射，从元素 s 开始,在元素 e 之前结束

*注意，这些 boolean 变元都是可选的。如果他们存在于 java6 中的方法，则润许指定是否包含端点，且这些方法会返回 NavigableXxx。如果 boolean 变元不存在，则方法返回 SortedSet 或 SortedMap。

109.与”导航”相关的重要方法

方法	描述
TreeSet.ceiling(e)	返回大于等于 e 的最小元素

TreeMap.ceilingKey(key)	返回大于等于 key 的最小键
TreeSet.highter(e)	返回大于 e 的最小元素
TreeMap.highterMap(key)	返回大于 key 的最小键
TreeSet.floor(e)	返回小于等于 e 的最大元素
TreeMap.floorKey(key)	返回小于等于 key 的最大键
TreeSet.lower(e)	返回小于 e 的最大元素
TreeMap.lowerKey(key)	返回小于 key 的最大键
TreeSet.pollFirst()	返回并删除第一项
TreeMap.pollFirstEntry()	返回并删除第一个键/值对
TreeSet.pollLast()	返回并删除最后一项
TreeMap.pollLastEntry()	返回并删除最后一个键/值对
TreeSet.descendingSet()	以反序返回 NavigableSet
TreeMap. descendingMap()	以反序返回 NavigableMap

110. 记住，当使用实现 MAP 的类是，用该映射的键的一部分的任何类都必须重写 hashCode() 和 equal()方法。

111.我们已经讨论过了许多关于自然顺序排序和使用 Comparator 排序的内容。应该牢记最后一条规则是：每次希望排序一个数组或一个集合时，内部的元素都必须是相互可比较的。换句话说，如果你有一个 Object[]，并将 Cat 和 Dog 对象放到其中，那么就不能排序它。

112.一定要知道在实践中如何解释如下表。对于考试可能会要求你根据特定的要求选择集合，这种要求被表达为一种场景。例如，如果需要维护和查找一个部件列表，他们由其唯一的字母数字序列号标识，其中的部件是 Part 类型的，这时应该使用哪种集合？如果改变要求，以便还能够按顺序、按它们的序列号打印出部件，你会改变答案吗？对于第一个问题，可以看出因为有了一个 Part 类，但是需要根据序列号查找对象，所以需要一个 Map。键将是 String 类型的序列号，而值是 Part 实例。默认选择应该是 HashMap，这是访问最快的 Map。但是，现在要求改变了，要按照部件的序列号顺序获得部件，则我们需要 TreeMap-它保持键的自然顺序。因为键是 String 类型，所以 String 的自然顺序将是标准的字母顺序。如果要求变了要记录最后访问的是哪个部件，则很可能需要 LinkedHashMap。但是由于 LinkedHashMap 不使用自然顺序(而是用最后访问的顺序)，所以，如果需要按照序列号列出部件，则必须显示地使用某种使用工具方法对该集合排序。

类	Map	Set	List	Ordered	sorted
HashMap	X			否	否
Hashtable	X			否	否
TreeMap	X			sorted	按照自然顺序或者自定义比较规则
LinkedHashMap	X			按照插入顺序或者最后	否

				的访问顺序	
HashSet		X		否	否
TreeSet		X		sorted	按照自然顺序或者自定义比较规则
LinkedHashSet		X		按照插入顺序	否
ArrayList			X	按照索引	否
Vector			X	按照索引	否
LinkedList			X	按照索引	否
PriorityQueue				sorted	按照”要求执行的任务”的顺序

113.LinkedList 中的迭代可能比 ArrayList 慢，但是当需要快速的插入和删除时，它是一个不错的选择。当需要快速迭代但不会做大量的插入和删除操作时，应选择 ArrayList 而不是 LinkedList。

114.NumberFormat 类的 format 会对超出指定位数的小数部分做四舍五入的操作.测试代码如下:

```
double d=987.123455d;
NumberFormat nf=NumberFormat.getInstance();
nf.setMaximumFractionDigits(5);
System.out.println(nf.format(d));
```

结果: 987.12346

115.当类实现 serializable 接口时，表示类可以被序列化。但如果类中的成员变量不能被序列化时。这个类还是不能被序列化。尝试序列化会抛出异常。如下所示:

```
class Vehicle{ }
class Wheels{   }
class   Car     extends   Vehicle implements Serializable { }
class   Ford    extends   Car   { }
class   Dodge   extends   Car {
    Wheels   w=new Wheels();
}
```

以上代码中：Ford 类可以被序列化而 Dodge 类则不能被序列化。

116.只有在写入数据时才需要调用 flush()方法。与读相关的类不具备这个方法。

117. 对于基本类型的包装类，在使用==和 equals()方法进行比较时结果是一样的。会自动进行卸装操作。并且基本类型必须相同。否则使用==编译时会出错，equals()方法返回 false.

118. 让一个类实现 serializable 不会有任何问题，即使其父类没有实现 serializable 也是如此。但是，当反序列化这样一个对象时，非序列化的父类必须运行它的构造函数。记住，不会在实现 serializable 的反序列化的类上运行构造函数。

119.Scanner 类的方法,hasNextXxx()的方法会测试下一个标记的值，但是不会实际获得该标记，也不会移动到源数据中的下一个标记。所有的 nextXxx()方法都执行两个功能：获得下一个标记，然后移动移动到下一个标记。

120.由于考试中频繁使用 System.out.println(),你可能会在关于大多数主题(包括正则表达式)的问题中看到使用转义序列的例子。记住，如果需要创建一个包含双引号或者反斜杠的字符串，则必须首先添加一个转义字符：

```
System.out.println(" \" \\");
```

这会输出:

```
" \
```

那么，如果需要在源数据中查找句点(.)，该怎么做呢？如果直接将一个句子放在正则表达式中，则会得到"任意字符"行为。那么，如果尝试使用"\."，则会如何呢？java 编译器会认为你正在尝试创建一个并不存在的转义序列。正确语法是:

```
String s="ab.cd.fg";
String[] tokens=s.split("\\.");
```

121.正则表达式查找的执行顺序是从左到右，只要在一个匹配中用到了源的一个字符，就不能在重复使用它。

122.主要 java.text 类和 java.util 类的实例创建

类	主要的实例创建选项
Util.Date	new Date() new Date(long millisencinds)
util.Calendar	Calendar.getInstance(); Calendar.getInstance(Locale)
Util.Locale	Locale.getDefault() new Locale(String language) new Locale(String language,String country)
text.DateFormat	DateFormat.getInstance() DateFormat.getDateInstance() DateFormat.getDateInstance(style) DateFormat.getDateInstance(style,Locale)

text.NumberFormat	NumberFormat.getInstance() NumberFormat.getInstance(Locale) NumberFormat.getNumberInstance() NumberFormat.getNumberInstance(Locale) NumberFormat.getCurrencyInstance() NumberFormat.getCurrencyInstance(Locale)

DateFormat.getInstance()：获取为日期和时间使用 SHORT 风格的默认日期/时间格式器。
DateFormat.getDateInstance()：获取日期格式器，该格式器具有默认语言环境的默认格式化风格。
NumberFormat.getInstance()和 NumberFormat.getNumberInstance()方法相同
DateFormat 和 NumberFormat 对象只有在实例化时才能进行地区设置。要小心那些试图改变现有实例的地区的代码----没有这样的方法存在。

123.处理日期和数字时常见的需求：

需求	步骤
获取当前时期和时间	1.创建一个 Date:Date d=new Date() 2.获得它的值:String s=d.toString();
获得一个对象，它允许在你的地区执行日期和时间计算	1. 创建一个 Calendar: Calendar c=Calendar.getInstance(); 2.使用 c.add(...) 和 c.roll(...)执行日期和时间操作
获取一个对象，它允许在不同的地区执行日期和时间计算	1. 创建一个 Locale: Locale loc=new Locale(language) 或者 Locale loc=new Locale(language,country) 2. 为该地区创建一个 Calendar: Calendar c=Calendar.getInstance(loc); 3.使用 c.add(...) 和 c.roll(...)执行日期和时间操作
获得一个对象，它允许执行日期和时间计算，然后对其进行格式化输出，用不同的日期风格用于不同的地区	1. 创建一个 Calendar: Calendar c=Calendar.getInstance(); 2 为每个地区创建一个 Locale: Locale loc=new Locale(language) 或者 Locale loc=new Locale(language,country) 3.将 Calendar 转换成 Date:Date d=c.getTime() 4.为每个 Locale 创建一个 DateFormat: DateFormat df= DateFormat.getDateInstance(style,loc) 5.使用 format()方法创建格式化日期:String s=df.format(d);
获取一个对象，允许跨越多个不同的地区格式化数字或货币	1. 为每个地区创建一个 Locale: Locale loc=new Locale(language)或者 Locale loc=new Locale(language,country) 2.创建一个 NumberFormat:NumberFormat nf=

	NumberFormat.getInstance(loc)或者 NumberFormat:NumberFormat nf= NumberFormat.getCurrencyInstance(loc) 3.使用 format()方法创建格式化输出:String s=nf.format(someNumber);

124.当反序列化一个序列化的实例时，构造函数不会运行，而且实例变量不会得到它们的最初值。如果一个序列化的类，但是其超类是非序列化的，那么从超类继承的任何实例变量，都将重置为它们在对象的原始构造期间所赋予的值。这是因为非序列化的类的构造函数会将运行。

125.java.io.Console 类

readLine()方法返回一个字符串，包含用户键入的任何内容。

rendPassword()方法并不返回一个字符串，它返回一个字符数组。原因：获得口令之后，就可以验证它，然后从内存完全删除。如果返回的是字符串，则它会在于内存中某处的 String 池中，黑客就能发现它。

126.迷你型 java.io API

java.io 类	扩展自	主要构造函数的变元	主要方法
File	Object	File,String String String,String	createNewFile() delete() exists() isFile() isDirectory() isFile() mkdir() list() renameTo()
FileWriter	Writer	File String	close() flush() write()
2BufferedWriter	Writer	Writer	close() flush() write() newLine()
PrintWriter	Writer	File(从 java5 起) String(从 java5 起) OutputStream Writer	close() flush() format() print() write() printf()

			println()
FileReader	Reader	File String	read()
BufferedReader	Reader	Reader	read() readLine()

127.数组有一个称作 length 的属性。而字符串则是 length()方法。

128.被重写的方法能抛出的检验异常相比，重写方法不能抛出比它更广的检验异常。但是，重写方法能够抛出 RuntimeException 异常，它可以不是由被重写的方法抛出。

129.当 assert 语句具有两个表达式时，第二个表达式必须返回一个值。

130. 断言的命令行开关

命令行例子	含义
java –ea java –enableassertions	启动断言
java –da java –disableassertions	禁用断言(Java6 的默认行为)
java –ea :com.foo.Bar	在类 com.foo.Bar 中启用断言
java –ea:com.foo..	在包 com.foo 以及它的任何子包中启用断言
Java –ea -dsa	通常启用断言，但在系统类中禁用
Java –ea –da:com.foo..	通常启用断言，但在包 com.foo 以及它的任何子包中禁用

131.使用 java6 将 assert 用作标识符或关键字时的编译代码

命令行	Assert 为标识符	Assert 为关键字
Javac –source 1.3 TestAsserts.java	代码通过编译，但有警告	编译失败
Javac –source 1.4 TestAsserts.java	编译失败	代码通过编译
Javac –source 1.5 TestAsserts.java	编译失败	代码通过编译
Javac –source 5 TestAsserts.java	编译失败	代码通过编译

Javac –source 1.6 TestAsserts.java	编译失败	代码通过编译
Javac –source 6 TestAsserts.java	编译失败	代码通过编译
Javac TestAsserts.java	编译失败	代码通过编译

132.常见异常的描述和来源（注意异常抛出的来源）

异常	描述	通常由谁抛出
ArrayIndexOutOfBoundsException	试图使用一个无效索引值(可能为负或超出了数组的大小)访问数组时抛出	JVM
ClassCastException	试图将一个引用变量强制转换为一个不能通过 IS-A 测试的类型时抛出	JVM
IllegalArgumentException	当方法接受到的变元格式不同于该方法需要的格式时抛出	以编程方式
IllegalStateException	当环境的状态与正在尝试的操作不匹配时抛出(如：使用一台已关闭的扫描仪)	以编程方式
NullPointerException	试图访问带有一个当前值为 null 的引用变量的对象时抛出	JVM
NumberFormatException	一个用于将 String 转换为数值的接受到一个不能转换的 String 是抛出	以编程方式
AssertionException	当一条语句的布尔测试返回 false 时抛出	以编程方式
ExceptionInInitializerError	试图初始化静态变量或静态初始化块时抛出	JVM
StackOverflowError	当方法递归层次过深时，通常会抛出这个异常(每次调用都会添加到栈中)	JVM
NoClassDefFoundError	由于命令行错误、路径问题，或者丢失了.class 文，使 JVM 找不到需要的类时，抛出这个异常	JVM

133.每个方法必须通过提供 catch 子句处理所有的检验异常，或者将每个未处理的检验异常列为一个抛出的异常。

134. 对于考试，不必了解包含在 Throwable 类中的任何方法，包括 Exception 和 Error。需要了解的是 Exception、Error、RuntimeException 和 Throwable 类型的异常都能够使用 throw 关键字抛出，并且都能够被捕获。

135.带标签的 continue 和 break 语句必须位于具有相同标签名称的循环内，否则代码将无法编译。

136.枚举在被进行变历史，values 返回的数组顺序就是枚举的顺序，枚举可以使用 equals()或==进行比较。

137.不要把布尔表达式中的=错认为==。下面的代码是合法的:

```
Boolean b=false;
If(b=true) {System.out.println(“b is true”);
}else{
    System.out.println(“b is false”);
}
```

输出的结果是:b is true.
用=代替==只对布尔变量有效。

138.字符能够用在比较运算中。当比较两个字符，或者将字符与数字比较时。Java 将字符的 Unicode 值当作数值进行比较。

140. 使用符合运算符时，=右边的表达式总是先求值。

```
Int x=2;
X*=2+5;
```

结果是:14.
相当于:x=x*(2+5);

141. 在考试的时候要小心，局部变量重复定义的错误。会发生编译异常。如:

```
Void go(int ouch){
    For(int ouch=3; ouch<10; ouch++){}
}
```

142.任何包装器类都不能从一种包装器加宽到另一种包装器！Byte 不能加宽到 Short，Short 不能加宽到 Long，等等。

143.记住，包装器引用变量可以为 null。这意味着必须关注那些看似会执行安全的基本操作，但可能会抛出 NullPointerException 异常代码。

```
Class Boxing2{
    Static Integer x;
    public static   void main(String[] args){
        doStuff(x);
    }
    Static void doStuff(int z){
        Int z2=5;
        System.out.println(z2+z);
    }
}
```

这段代码能通过编译，但是当它试图调用 doStuff(x)时，JVM 会抛出 NullPointerException 异常，因为 x 不能引用一个 Integer 对象，因此没有要拆箱的值。

144.字面型整数总是 int，但更重要的是，一个占用 int 或更短长度的表达式的结果将总是一个 int。换句话说，如果将两个 byte 加到一起将得到一个 int—即使这两个 byte 都很小；int 乘以 short 将得到 int；short 除以 byte 将得到 int。如下:

```
Byte a=3;
Byte b=8;
Byte c=a+b; //not complie. Have problem!
```

145.字符实际上只是一个 16 位无符号整数。只要它位于无符号 16 位数值的范围(小于或等于 65535)之内。以下语句都是合法的:

```
Char a=0x892;
Char b=982;
Char c=(char)70000;
Char d=(char)-98;
```

以下是非法的:

```
Char e=-29;
Char f=70000;
```

146.

```
System.out.println(3.0 / 0);          // Infinity
System.out.println(-3.0 / 0);         // -Infinity
System.out.println(0.0 / 0.00);       // NaN

System.out.println(3.0 % 0);          // NaN
System.out.println(-3.0 % 0);         // NaN
```

```
System.out.println(0.0 % 0.00);      // NaN
System.out.println(1/0);      // ArithmeticException
```

注意：整数和浮点数的不同结果

147.

```
byte x = -64;
byte y = -6;
System.out.println(x/y + " " + x%y);    // 10 -4
```

取模运算中，余数的正负号与左操作数一致。

148.

```
        int a = 9;
      a += (a = 3);
      System.out.println(a);    // 12
```

算术运算从左向右，赋值运算从右向左。

149.思考画图题：java 参数传递是值传递。

```
class ValHold{
        public int i = 10;
}
public class ObParm{
        public void amethod(){
                ValHold v = new ValHold();
                another(v);
                System.out.println(v.i);
        }
        public void another(ValHold v){
                v.i = 20;
                ValHold vh = new ValHold();
                v =vh;
                System.out.println(v.i);
        }
        public static void main(String[] argv){
                ObParm o = new ObParm();
                o.amethod();
        }
}
```

150.

```
class Base {
    static final int S;
    static float M;
    S=10;
    M=10.3f;
    Base(){}
}                           //compile error!
class Base {
    final int S;
    static float M;
    static final int S2;
    Base(){
        S=10;           //常量（不能是静态）可以在构造函数中赋值
        S2=10;          //error
        M=10.3f;        //ok，可以给静态变量赋值
        final int n=10; //ok, 构造方法中可以有定义局部常量
        static int i=5;  //compile error,构造方法中不能定义静态常量或变量
    }
}
```

构造方法中能做的：
1．给类变量（包括 static）赋值
2．定义局部非静态变量或常量

构造方法中不能做的：
1． 给类静态常量赋值
2． 定义局部静态常量或变量

任何非静态方法中都不能定义静态变量！构造函数也是

151. main() can be declared final. (OK)

152. String a="\uD7AF";// *(?)合法*，String b="\u000a";//*(new line)不合法*，String c="\n";//*合法*。

153. a = b ? c : d ? e : f //*執行順序為由右向左(d ? e : f 會先被執行)*

int x=4;println((x>4)?99.99:8);➔8.0(若 99.99 是 99，則印出 8)

154. do {

} while (condition)**;**

- condition 須為 boolean type。
- 至少會執行一次。

注意不要遺忘 while 之後的分號。

155. 内部类的理解

Meber Class：

```
class InnerSuper { protected int i = 1; }
 class OuterSuper { protected int i = 2; }
 public class Test extends OuterSuper {
       private int i = 3;
       public class Inner extends InnerSuper {
            private int i = 4;
            public void f() {
                int i = 5;
                System.out.print(i);                    // 5
                System.out.print(this.i);             // 4
                System.out.print(super.i);           // 1
                System.out.print(Test.this.i);    // 3
                System.out.print(Test.super.i); // 2
            }
      }
      public static void main(String[] args) {
            Test t = new Test();
            Inner inn = t.new Inner();
            inn.f();
      }
}
 class A {
      public static void main(String[] args) {
            Test.Inner inn = new Test().new Inner();
            inn.f();
      }
}
```

Local Class：

```
 class InnerSuper { protected int x = 1; }
 class OuterSuper { protected int y = 2; }
 interface ForPolymor { void f(); }
 class Test extends OuterSuper {
      private int z = 3;
      public ForPolymorphism g() {
           final int i = 4;
           int j = 5;
           class Inner extends InnerSuper implements ForPolymor {
              int k = 6;
```

```
            public void f() {
                int m = 7;
                System.out.print(x);     // 1
                System.out.print(y);     // 2
                System.out.print(z);     // 3
                System.out.print(i);     // 4
                //System.out.print(j);   // compile error
                System.out.print(k);     // 6
                System.out.print(m);     // 7
            }
        }
        Inner in = new Inner();  //create Inner class 的 object
        in.f();    //呼叫 Inner class 的 method
        return in;
    }
    public static void main(String[] args) {
        Test t = new Test();
        //Inner in = new Inner(); in.f(); //Inner is out of scope
        ForPolymor fp = t.g();
        fp.f();
    }
}
```

156.

- finally block 的動作會蓋過前面 try 或 catch block 的動作。即 finally block 的 return 或 throw statement 會蓋掉 try 或 catch block 的 return 或 throw statement。

157. The Math Class:

- 注意：是一個 final class，故不可被繼承，且所有的 mebers 都是 static members。
- 只有 private constructor，故無法 create 一個 Math object(也不須 create Math object)
- 不會 throws 任何的 exception。
- 對負數做開根號 sqrt()所得結果為 NaN。

Long、Ingeger 的最小絕對值是負的。

简单记：

(int、float、long、double)	abs、max、min
奇怪的	double random()、 long round(double a)、int round(float a)
只有 double	剩下的其他 method

158. The Integer's parseInt() method throws a NumberFormatException, but since it's a runtime exception, it doesn't need to be handled to compile and run.

159.枚举内可以定义字符。枚举有 ordinal()方法，返回枚举所在的位置(从 0 开始记起)。

160.考题中如果代码出现 import 导包时，注意两个类方法的修饰符。

161.考试时遇到 map 集合时，注意 map 没有 add（）方法。
162.考试时，如果在 Thred 题中遇到需要同步的代码，先看看代码中是否有 wait 出现，如果有则去查看是否有 notify 或 notifyAll 方法。Wait 和 notify 所在的同步块有相同的对象锁。

163.对于封装、耦合、内聚的说法：

41

What is true? (Choose all that apply.)

- A A class must be well encapsulated in order to be highly cohesive.
- B If two classes are NOT each highly cohesive, they can't be loosely coupled to each other.
- C Good encapsulation helps promote loose coupling.
- D High cohesion allows you to change a method's implementation without having to change its API.
- E Tight coupling helps promote high cohesion.

B 是错误的，because loose coupling describes a situation in which classes know only about each other's APIs, and nothing else.
D is also wrong, because this is the description for encapsulation

164.关于 java 命令行的试题：

68

Given that the MusicPlayer project is on a UNIX system and consists of the following files:
/mp/player/MusicPlayer.java
/mp/classes/player/MusicPlayer.class
/mp/jars/mp.jar
Inside mp.jar file the structure is:
player/MusicPlayer.java
player/MusicPlayer.class
You are currently in the directory
/mp
and the CLASSPATH is set to
/mp/jars
What command(s) can you use to invoke the class player.MusicPlayer? (Choose all that apply.)

- A java player.MusicPlayer
- B java player/classes/MusicPlayer.class
- C java -cp player MusicPlayer
- D java -cp classes MusicPlayer
- E java -cp classes player.MusicPlayer
- F java -cp /mp/classes/player MusicPlayer
- G java -cp jars/cp.jar MusicPlayer
- H java -cp /mp/jars/cp.jar player.MusicPlayer

E and H are correct. The location of the .java file is irrelevant; we need to specify where to find one of the .class files. The CLASSPATH is no good to us here since it

doesn't include any directory or jar file which directly contains player/MusicPlayer.class. The classpath must be set with the -cp option to include either /mp/classes or /mp/jars/mp.jar (so we can omit the initial /mp from either one since we're already starting from that directory).
A and B are wrong because they have no -cp (and the CLASSPATH is invalid). C and F are wrong because they have the wrong classpath. D, E, G, and H have the correct classpath, but D and G are wrong because the final argument must be player.MusicPlayer (we can't omit the package here).

165.wait 方法有重载时间的方法。而 notify 则没有这样的重载方法。

166.在 switch 判断的子句中，case 中必须是常量，final 修饰的变量可以使用。其他的变量会出现编译错误.

```
final static int x=1;
static int y=2;
public static void main(String[] args){
    for (int i = 0; i < 3; i++) {
        switch(i){
        case y:;
        case x:;
        }
    }

}
```

167.hashcode()和 equals()方法

52

Given:

```
class SortOf {
  String name;
  int bal;
  String code;
  short rate;
  public int hashCode() {
    return (code.length() * bal);
  }
  public boolean equals(Object o) {
    // insert code here
  }
}
```

Which of the following will fulfill the equals() and hashCode() contracts for this class?
(Choose all that apply.)

- [ ] A return ((SortOf)o).bal == this.bal;
- [ ] B return ((SortOf)o).code.length() == this.code.length();
- [x] C return ((SortOf)o).code.length() * ((SortOf)o).bal == this.code.length() * this.bal;
- [x] D return ((SortOf)o).code.length() * ((SortOf)o).bal * ((SortOf)o).rate == this.code.length() * this.bal * this.rate;

EXPLANATION:
C and D are correct. The equals() algorithm must be at least as precise in defining what "meaningfully equivalent" means as the hashCode() method is.
A and B are incorrect because these equals() implementations would allow instances to be equal that hashCode() would not see as equal.

168.finally 块在异常之前执行：

17

Given:

```
1.  public class MyProgram {
2.     public static void throwit() {
3.       throw new RuntimeException();
4.     }
5.     public static void main(String args[]){
6.       try {
7.          System.out.println("Hello world ");
8.          throwit();
9.          System.out.println("Done with try block ");
10.      }
11.      finally {
12.         System.out.println("Finally executing ");
13.      }
14.    }
15. }
```

Which answer most closely indicates the behavior of the program?

- ○ A The program will not compile.
- ○ B The program will print Hello world, then will print that a RuntimeException has occurred, then will print Done with try block, and then will print Finally executing.
- ○ C The program will print Hello world, then will print that a RuntimeException has occurred, and then will print Finally executing.
- ◉ D The program will print Hello world, then will print Finally executing, then will print that a RuntimeException has occurred.

EXPLANATION:
D is correct. Once the program throws a RuntimeException (in the throwit() method) that is not caught, the finally block will be executed and the program will be terminated. If a method does not handle an exception, the finally block is executed before the exception is propagated.
A, B, and C are incorrect based on the program logic just described.

169. Comparable<T>需要实现的接口方法是 **compareTo**(T o)。Comparator<T>需要实现的接口方法是 **compare**(T o1, T o2)。

170.String 类中没有 inster 方法。考试时需要注意。

171.变量只能包含：字符、数字、下划线和$。但开头只能是字符、下划线、$。

172.assert 断言的第二个参数如果是方法是，则方法必须要有返回值:

16

Given:

```
1.  public class Test {
2.    public static int y;
3.    public static int foo(int x) {
4.      System.out.print("foo ");
5.      return y = x;
6.    }
7.    public static void bar(int z) {
8.      System.out.print("bar ");
9.      y = z;
10.   }
11.   public static void main(String [] args ) {
12.     int t = 2;
13.     assert t < 4 : bar(7);
14.     assert t > 1 : foo(8);
15.     System.out.println("done ");
16.   }
17. }
```

What is the result?

- A bar
- B bar done
- C foo done
- D bar foo done
- E Compilation fails.

EXPLANATION:

E is correct. The bar() method returns void. It is a perfectly acceptable method, but because it returns void it cannot be used in an assert statement. Thus, line 13 will not compile.

A, B, C, and D are incorrect based on the program logic just described.

173.

36

Assuming that an interrupted exception has NOT been thrown and that "aLiveThread" is a runnable thread, which three guarantee that a thread will leave the running state?

- [ ] A yield()
- [x] B wait()
- [ ] C notify()
- [ ] D notifyAll()
- [x] E sleep(1000)
- [x] F aLiveThread.join()
- [ ] G Thread.killThread()

EXPLANATION:

B, E, and F are correct. B is correct because wait() always causes the current thread to go into the object's wait pool. E is correct because sleep() will always pause the currently running thread for at least the duration specified in the sleep argument (unless an interrupted exception is thrown). F is correct because, assuming that the thread you're calling join() on is alive, the thread calling join() will immediately block until the thread you're calling join() on is no longer alive.

A is wrong, but tempting. The yield() method is not guaranteed to cause a thread to leave the running state, although if there are runnable threads of the same priority as the currently running thread, then the current thread will probably leave the running state. C and D are incorrect because they don't cause the thread invoking them to leave the running state. G is wrong because there's no such method.

174.在考试中如果遇到了静态导入，需要注意导入的类是否正确.静态导入必须要指明确是哪个方法。

63

Given:

```
import static java.lang.System;
class _ {
  static public void main(String... __A_V_) {
    String $ = "";
    for(int x=0; ++x < __A_V_.length; )
      $ += __A_V_[x];
    out.println($);
  }
}
```

With the command line:

java _ - A .

What is the result?

- ○ A -A
- ○ B A.
- ○ C -A.
- ○ D _A.
- ○ E _-A.
- ◉ F Compilation fails.
- ○ G An exception is thrown at runtime.

EXPLANATION:

F is correct. The only thing wrong with this code is that the static import did not specify the member to be imported. This question is using valid (but weird) identifiers, var-args in main, and pre-incrementing logic.

A, B, C, D, E, and G are incorrect based on the preceding explanation.

175.静态方法可以实现重载。特别是在有继承关系的两个类中。在重载方法中，包装类参数>动态参数的方法。

23

Given:

```
class High {
  public static String go(int... x) { return "hi "; }
}
class Low extends High {
  public static String go(Integer x, Integer y) { return "low "; }
  public static void main(String[] ack) {
    System.out.print(go(9,27));
    System.out.println(go(81));
  }
}
```

What is the result?

- A hi hi
- B hi low
- C low hi
- D Compilation fails.
- E An exception is thrown at runtime.

176.能修饰类的关键字:abstract、final。接口不能定义 static 方法。

177.

73

To create an instance of a new, Java 6 Console, class(es) from which package(s) must be used? (Choose all that apply.)

- [x] A java.io
- [ ] B ava.awt
- [x] C java.lang
- [ ] D java.text
- [ ] E java.util
- [ ] F javax.swing

EXPLANATION:
A and C are correct. A line of code such as Console c = System.console(); creates a new instance of java.io.Console, and the System class is in the java.lang package.
B, D, E, and F are incorrect based on the above.

178.对于断言的使用：

18

Which are true? (Choose all that apply.)

- [ ] A It is not good practice to place assertions where you think execution should never reach.
- [ ] B It is sometimes appropriate to call getters and setters from assertions.
- [x] C Use assertions to verify the arguments of private methods.
- [x] D Assertions can be disabled for a particular class.
- [ ] E Never throw an AssertionError explicitly.

179.泛型的错误认识：

60

Given:

```
class Food { }
class Fruit extends Food { }
class Apple extends Fruit { }
// insert code here
  public static void main(String[] munch) {
    Pie<Fruit> p = new Pie<Fruit>();
  }
   }
```

Which inserted at // insert code here, will compile? (Choose all that apply.)

- [x] A class Pie<T extends Food> {
- [x] B class Pie<T extends Fruit> {
- [ ] C class Pie<T extends Apple> {
- [ ] D class Pie<T extends Pie> {
- [ ] E class Pie<T super Apple> {
- [x] F class Pie<T> {

180.对 File 的理解：

28

What is true? (Choose all that apply.)

- [x] A A java.io.File object can represent a file on a file system.
- [x] B A java.io.File object can represent a directory on a file system.
- [x] C The File.delete() method can delete directories only if they are empty.
- [x] D The File.renameTo() method can rename directories even if they are NOT empty.

181.对于反序列化的理解:

29

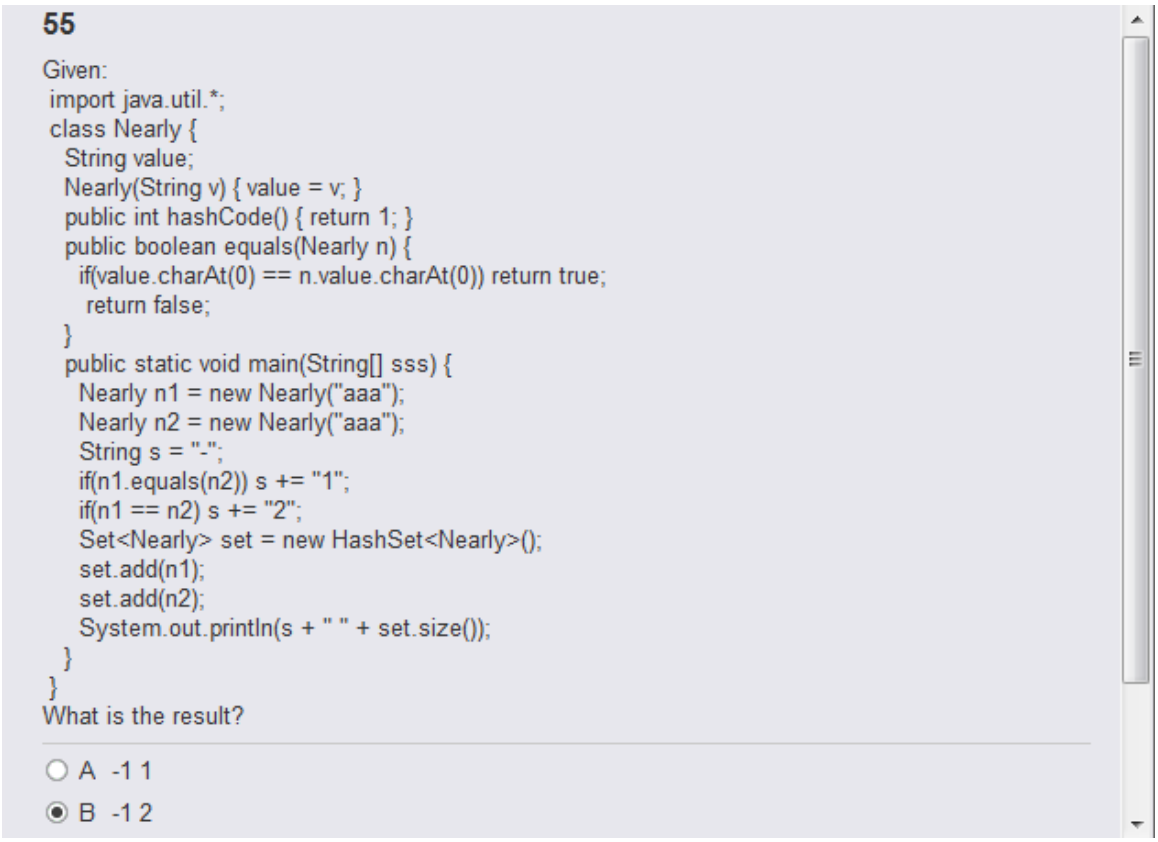
Given a class X with an instance variable:
transient int readCount;
If you want to serialize and deserialize an instance of class X, AND retain readCount's value through the serialization and deserialization process, what is true about the method you must implement on the deserialization side of the process? (Choose all that apply.)

- [ ] A It must be marked public void.
- [x] B It must be marked private void.
- [ ] C It must be marked public boolean.
- [ ] D It must be marked private boolean.
- [ ] E It takes an object of type InputStream.
- [x] F If takes an object of type ObjectInputStream.

需要去实现：The method you must implement to do custom deserialization is: private void

readObject(ObjectInputStream is) throws IOException, ClassNotFoundException

182.注意 Object 的 equals 方法的参数是 Object.如果子类的 equals 方法不是 Object 参数则不是重写父类的方法。判断对象是否相等的方法还是 Object 的 equals 方法。



55

Given:

```
import java.util.*;
class Nearly {
  String value;
  Nearly(String v) { value = v; }
  public int hashCode() { return 1; }
  public boolean equals(Nearly n) {
    if(value.charAt(0) == n.value.charAt(0)) return true;
     return false;
  }
  public static void main(String[] sss) {
    Nearly n1 = new Nearly("aaa");
    Nearly n2 = new Nearly("aaa");
    String s = "-";
    if(n1.equals(n2)) s += "1";
    if(n1 == n2) s += "2";
    Set<Nearly> set = new HashSet<Nearly>();
    set.add(n1);
    set.add(n2);
    System.out.println(s + " " + set.size());
  }
}
```

What is the result?

- A -1 1
- B -1 2

Two different equals() methods are invoked, because the equals() method shown in the code doesn't properly override Object.equals(), which takes an Object.

183.强制把父类转换成子类不会出现编译异常，但要注意会出现运行时异常.

45

Given:

```
class X { void go() { System.out.print("x "); } }
class Y extends X { void go() { System.out.print("y "); } }
class Z extends X { void go() { System.out.print("z "); } }
class Chrome2 {
  public static void main(String [] args) {
    X z = new Z();
    X y = new Y();
    Z y2 = (Z)y;
    z.go();
    y.go();
    y2.go();
  }
}
```

What is the result?

- ○ A x x x
- ○ B x x z
- ○ C z y y
- ○ D z y z
- ○ E Compilation fails.
- ◉ F An exception is thrown at runtime.

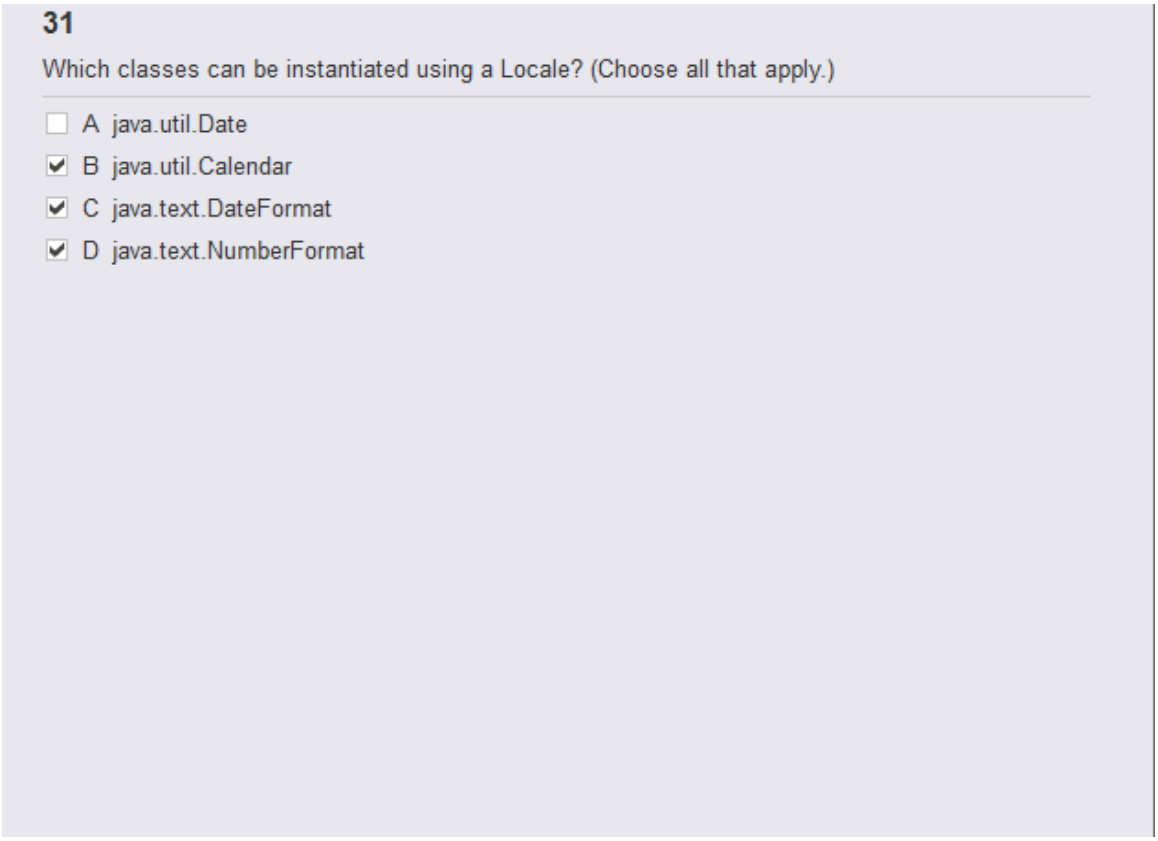
184.对于封装的理解：

43

Which are true? (Choose all that apply.)

- ☑ A Encapsulation limits the consequences of change.
- ☑ B Encapsulation allows corrections to a class with minimal impact on users of that class.
- ☑ C Encapsulation makes it easier to reuse classes.
- ☑ D Encapsulation results in better testing and higher reliability.
- ☐ E Encapsulation ensures that member variables are readily accessible.
- ☐ F Encapsulation means that methods are all marked public.
- ☐ G Encapsulation means that classes are members of packages.

185.对于日期函数构造的记忆：



186.正则表达式：
预定义字符类：
\s： 空白字符
\w: [a-z A-Z 0-9]
\d :[0-9]

187.对于比较类的描述：

54

Which are true? (Choose all that apply.)

- [ ] A You can use java.lang.Comparable and java.util.Comparator to sort collections whose elements are of any valid Java type, as long as all of the collection's elements are of the same class.
- [ ] B The java.lang.Object class implements the java.lang.Comparable interface.
- [x] C Many commonly used classes in the Java API (such as String, Integer, Date, and so on) implement the java.lang.Comparable interface.
- [ ] D If your class implements java.lang.Comparable but you don't explicitly override Comparable's method, collections containing elements of your class will be sorted in natural order by default.
- [x] E When using the java.util.Comparator interface, you will typically create a separate class for every different sort sequence you want to implement.
- [ ] F The java.util.Comparator interface's method can take either one or two arguments.
- [ ] G The binarySearch(), reverse(), and reverseOrder() methods in the java.util.Collections class all require that the collection is sorted before the method can be invoked successfully.

189.看看的

52

Which are true? (Choose all that apply.)

- [ ] A Has-a relationships are, by definition, well encapsulated.
- [ ] B A covariant return requires the existence of a has-a relationship.
- [x] C Overriding requires the existence of a is-a relationship.
- [x] D In some cases, is-a relationships are used in the process of autoboxing.
- [ ] E Has-a relationships are always defined with instance variables.

190.关于 java 命令的理解:

66

Which are true? (Choose all that apply.)

- [ ] A Using the -d command-line argument instructs the compiler to place the .class file in the same directory as the .java file.
- [ ] B Using the -D command-line argument instructs the compiler to place the .class file in the directory specified after the -D argument.
- [x] C The -classpath command-line argument can be used with both the java and javac commands.
- [ ] D When the compiler gets the command-line argument that instructs it to construct any missing directories it needs, a destination directory must NOT be included in the command line.
- [ ] E When using the java command, you must specify one or more (already compiled) files to execute.
- [x] F When using the java command, only files ending in .class can be executed.

191.File 操作时，close()也需要进行 try,catch 进行异常处理。

21

Given:

```
1.  import java.io.*;
2.  public class Test {
3.     public static void main(String [] args){
4.        FileInputStream in = null;
5.        try {
6.           in = new FileInputStream("test.txt");
7.           int x = in.read();
8.        }
9.        catch(IOException io) {
10.           System.out.println("IO Error.");
11.       }
12.       finally {
13.          in.close();
14.       }
15.    }
16. }
```

And given that all methods of class FileInputStream throw an IOException, which of the following is true? (Choose one.)

- ○ A This program will compile successfully.
- ○ B This program fails to compile due to an error at line 4.
- ○ C This program fails to compile due to an error at line 6.
- ○ D This program fails to compile due to an error at line 9.
- ◉ E This program fails to compile due to an error at line 13.

192.多态只使用于方法。对于类的属性不使用。注意下面的试题：

48

Given:

```
class Car2 {
   static int i1 = 5;
  int i2 = 6;
  public static void m1() { System.out.print(i1); }
  public void m2() { System.out.print(i2); }
}
class Mini2 extends Car2 {
  static int i1 = 7;
  int i2 = 8;
  public static void m1() { System.out.print(i1); }
  public void m2() { System.out.print(i2); }
  public static void main(String[] args) {
    Car2 c = new Mini2();
    c.m1();
    System.out.print(" " + c.i1 + c.i2);
  }
}
```

What is the result?

- (●) A 5 56
- ( ) B 5 58
- ( ) C 5 78
- ( ) D 7 56
- ( ) E 7 58

Remember that polymorphism applies only to instance methods, not to static methods, and not to either static or instance variables.

193.对于命令行的理解：

66

Which are true? (Choose all that apply.)

- [ ] A Using the -d command-line argument instructs the compiler to place the .class file in the same directory as the .java file.
- [ ] B Using the -D command-line argument instructs the compiler to place the .class file in the directory specified after the -D argument.
- [x] C The -classpath command-line argument can be used with both the java and javac commands.
- [ ] D When the compiler gets the command-line argument that instructs it to construct any missing directories it needs, a destination directory must NOT be included in the command line.
- [ ] E When using the java command, you must specify one or more (already compiled) files to execute.
- [x] F When using the java command, only files ending in .class can be executed.

194. 对于 Arrays.asList 的理解。

public static <T> List<T> **asList**(T... a)

返回一个受指定数组支持的固定大小的列表。（对返回列表的更改会“直接写”到数组。）此方法同 Collection.toArray() 一起，充当了基于数组的 API 与基于 collection 的 API 之间的桥梁。返回的列表是可序列化的，并且实现了 RandomAccess。

此方法还提供了一个创建固定长度的列表的便捷方法，该列表被初始化为包含多个元素：

```
List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");
```

参数：

a - 支持列表的数组。

返回：

指定数组的列表视图。

195.重点要知道 floor(), higher(), and ceiling() return single elements, not subsets.

74

Given:

```
3. import java.util.*;
4. public class Inca {
5.   public static void main(String[] args) {
6.     TreeSet<Integer> stuff = new TreeSet<Integer>();
7.     stuff.add(2); stuff.add(4); stuff.add(6); stuff.add(8);
8.     // insert code here
9.   }
10. }
```

Which code fragment(s) added independently at line 8, produce the output [2, 4]? (Choose all that apply.)

- [ ] A System.out.println(stuff.floor(6));
- [ ] B System.out.println(stuff.higher(6));
- [ ] C System.out.println(stuff.ceiling(6));
- [x] D System.out.println(stuff.headSet(6));
- [ ] E System.out.println(stuff.tailSet(6));
- [ ] F Compilation fails regardless of which code is inserted.

196.Throwable is not a subclass of Exception.

197.强转类型不会出现编译失败而是会出现运行时异常：

45

Given:

```
class X { void go() { System.out.print("x "); } }
class Y extends X { void go() { System.out.print("y "); } }
class Z extends X { void go() { System.out.print("z "); } }
class Chrome2 {
  public static void main(String [] args) {
    X z = new Z();
    X y = new Y();
    Z y2 = (Z)y;
    z.go();
    y.go();
    y2.go();
  }
}
```

What is the result?

- ○ A x x x
- ○ B x x z
- ○ C z y y
- ○ D z y z
- ○ E Compilation fails.
- ◉ F An exception is thrown at runtime.

198.

6

Given:

```
1.  public class Test {
2.     public static void main(String [] args) {
3.        short [][] b = new short [4][4];
4.        short [][] big = new short [2][2];
5.        short b3 = 8;
6.        short b2 [][][][] = new short [2][3][2][2];
7.        // insert code here
8.     }
9.  }
```

Which of the following lines of code could be inserted at line 7 and still allow the code to compile? (Choose all that apply.)

- ☑ A b2[1][1] = big;
- ☑ B b[1][0] = b3;
- ☐ C b2[0][1][1] = b;
- ☐ D b2[0][2][1] = b[1][0];
- ☑ E b2[1][1][0][1] = b[1][0];
- ☑ F b2[1][1] = b;

注意最后一个答案：虽然 b 的范围比 b2[1][1]大。但是编译是通过的。没有错误！

199.

2

Which are legitimate package declarations for a source file? (Choose all that apply.)

- [x] A package utils.mystuff.com;
- [ ] B package com.mystuff.*;
- [x] C package com.mystuff.utils;
- [ ] D package com.mystuff.utils; package com.yourstuff.other;

注意：一个类只能有一个包定义。

200.注意一个类的 main 方法在调用自己类的静态内部类是的写法：以下的代码没有任何异常。

40

Given the following:

```
public class SyncTest {
    private static Foo foo = new Foo();
    static class Increaser extends Thread {
        public void run() {
            foo.increase(20);
        }
    }
    public static void main(String [] args) {
        new Increaser().start();
        new Increaser().start();
        new Increaser().start();
    }
}
class Foo {
    private int data = 23;
    public void increase(int amt) {
        data = data + amt;
        System.out.println("Data is: " + data);
    }
}
```

Assuming that data must be protected from corruption, what (if anything) can you add to the preceding code to ensure the integrity of the data?

- ○ A Synchronize the run() method.
- ○ B Wrap a synchronize(this) around the call to f.increase().

201.注意 Map 的 subMap 的用法。如果有两个子类则改变其中一个子类，另一个子类也会改变：答案是:E 6 4 4

75

Given the proper import statements, and:

```
11. TreeMap<String, String> tmap;
12. tmap = new TreeMap<String, String>();
13. tmap.put("a", "apple"); tmap.put("b", "banana");
14. tmap.put("c", "cantalope"); tmap.put("d", "date");
15. NavigableMap<String, String> nmap;
16. NavigableMap<String, String> nmap2;
17. nmap = tmap.subMap("b", true, "c", true);
18. nmap2 = tmap.subMap("b", true, "c", true);
19. tmap.put("b2", "blueberry");
20. nmap.put("b3", "brownie");
21. System.out.println(tmap.size()+" "+nmap.size()+" "+nmap2.size());
```

Which are true? (Choose all that apply.)

- [ ] A The output is 5 3 2
- [ ] B The output is 5 4 2
- [ ] C The output is 5 4 3
- [x] D The output is 6 4 3
- [ ] E The output is 6 4 4
- [ ] F Compilation fails.

202.注意 abstract 修饰类时，只能放在 class 修饰符的前面：答案 D 是错误的。

71

Which declare a compilable abstract class? (Choose all that apply.)

- [x] A public abstract class Canine { public void speak(); }
- [x] B public abstract class Canine { public void speak() { } }
- [ ] C public class Canine { public abstract void speak(); }
- [x] D public class Canine abstract { public abstract void speak(); }

203.关于泛型方法的题：答案是:A B D E

57

Given:

```
import java.util.*;
public class BackLister {
   // INSERT HERE
   {
      List<T> output = new LinkedList<T>();
      for (T t : input)
         output.add(0, t);
      return output;
   }
}
```

Which of the following can be inserted at // INSERT HERE to compile and run without error?

- [x] A public static <T> List<T> backwards(List<T> input)
- [ ] B public static <T> List<T> backwards(List<? extends T> input)
- [x] C public static <T> List<T> backwards(List<? super T> input)
- [ ] D public static <T> List<? extends T> backwards(List<T> input)
- [ ] E public static <T> List<? super T> backwards(List<T> input)
- [ ] F public static <? extends T> List<T> backwards(List<T> input)
- [ ] G public static <? super T> List<T> backwards(List<T> input)

204.关于线程优先级常量的问题：

字段摘要

static int	**MAX_PRIORITY** 线程可以具有的最高优先级。
static int	**MIN_PRIORITY** 线程可以具有的最低优先级。
static int	**NORM_PRIORITY** 分配给线程的默认优先级。

35

Assume you create a program and one of your threads (called backgroundThread) does some lengthy numerical processing. What would be the proper way of setting its priority to try to get the rest of the system to be very responsive while the thread is running? (Choose all that apply.)

- [x] A backgroundThread.setPriority(Thread.LOW_PRIORITY);
- [x] B backgroundThread.setPriority(Thread.MAX_PRIORITY);
- [ ] C backgroundThread.setPriority(1);
- [ ] D backgroundThread.setPriority(Thread.NO_PRIORITY);
- [ ] E backgroundThread.setPriority(Thread.MIN_PRIORITY);
- [ ] F backgroundThread.setPriority(Thread.NORM_PRIORITY);
- [ ] G backgroundThread.setPriority(10);

205.关于接口中的变量和方法修饰符需要注意：

变量：public、static、final 修饰。或者什么都不加。

方法: public、abstract 修饰。或者什么都不加。

4

Which are valid declarations within an interface? (Choose all that apply.)

- [x] A static long shanks = 343;
- [x] B protected static short timer = 22;
- [ ] C private short hop = 23;
- [ ] D final int stufflt(short top);
- [x] E public void doMore(long bow);
- [x] F static byte doMore(double trouble);

上题目正确答案只有：A E

206.静态方法中不能使用 super 关键字.

47

Given:

```
class AlternateFuel {
  int getRating() { return 42; }
  static int getRating2() { return 43; }
}
class BioDiesel extends AlternateFuel {
  public static void main(String[] args) {
    new BioDiesel().go();
    System.out.print(super.getRating2());   // #1
  }
  void go() {
    System.out.print(super.getRating());    // #2
  }
}
```

What is the result?

- ( ) A 4243
- (•) B 4342
- ( ) C Compilation fails due only to an error on line #1.
- ( ) D Compilation fails due only to an error on line #2.
- ( ) E Compilation fails due to errors on both lines #1 and #2.
- ( ) F An exception is thrown at runtime.

答案是：C

207. Console 类的 readLine 方法解释如下：

readLine

```
public String readLine(String fmt,
                       Object... args)
```

提供一个格式化提示，然后从控制台读取单行文本。

参数：

fmt - 格式字符串语法中描述的格式字符串。
args - 格式字符串中的格式说明符引用的参数。如果参数多于格式说明符，则忽略额外的参数。参数的最大数量受到 Java 虚拟机规范定义的 Java 数组最大维数的限制。

返回：

包含从控制台读取的行的字符串，该字符串不包含任何行终止符；如果已到达流的末尾，则返回 null。

抛出：

IllegalFormatException - 如果格式字符串包含非法语法、与给定参数不兼容的格式说明符、对给定格式字符串而言不够的参数或其他非法条件。有关所有可能的格式错误的规范，请参阅 formatter 类规范的详细信息部分。
IOError - 如果发生 I/O 错误。

如果控制台没有任何输入会出现 java.lang.NullPointerException

73

Given:

```
3. import java.io.*;
4. public class TestThis implements Runnable {
5.   public static void main(String[] args) {
6.     Thread t = new Thread(new TestThis());
7.     t.start();
8.     Thread t2 = new Thread(new TestThis());
9.     t2.start();
10.    Console c = System.console();
11.    String u = c.readLine("%s", "m ");
12.    System.out.print(u + " ");
13.  }
14.  public void run() {
15.    Console c = System.console();
16.    String u = c.readLine("%s", "t ");
17.    System.out.print(u + " ");
18.  }
19. }
```

What is the result? (Choose all that apply.)

- [x] A Compilation fails.
- [ ] B A NullPointerException is never thrown.
- [ ] C A NullPointerException is always thrown.
- [ ] D A NullPointerException is sometimes thrown.

答案是:C

208.注意 TreeSet 集合会排序

55

Given:

```
import java.util.*;
public class Numbers {
    private Set<Integer> numbers = new TreeSet<Integer>();
    public Numbers(int... nums) {
        for (int n : nums)
            numbers.add(n);
    }
    public Numbers negate() {
        Numbers negatives = new Numbers();
        for (int n : numbers)
            negatives.numbers.add(-n);
        return negatives;
    }
    public void show() {
        for (int n : numbers)
            System.out.print(n + " ");
    }
    public static void main(String[] args) {
        new Numbers(1, 3, -5).negate().show();
    }
}
```

What is the result?

正确答案是: -3 -1 5

209 。

58

Given:

```
import java.util.*;
public class Mangler {
    public static <K, V> Map<V, K> mangle(Map<K, V> in) {
        Map<V, K> out = new HashMap<V, K>();
        for (Map.Entry<K, V> entry : in.entrySet())
            out.put(entry.getValue(), entry.getKey());
        return out;
    }
    public static void main(String[] args) {
        Map m1 = new HashMap();
        m1.put("a", 1);
        m1.put("b", 2);
        Map m2 = mangle(m1);
        System.out.println(m2.get("a") + " " + m2.get(2));
    }
}
```

What is the result?

答案是 `null b`

```
out.put(entry.getValue(), entry.getKey());
```

把键值调换了。

210.注意线程的 join 方法：

38

Given:

```
public class ThreadTest  {
  class InnerRun implements Runnable {
    public void run() {
      for (int x = 0; x < 100; x++) {
        try {
          Thread.sleep(5);
        } catch (Exception e) { }
        System.out.print("Ren ");
      }
    }
}
class InnerRunTwo implements Runnable {
  Thread other;
  InnerRunTwo(Thread t) { other = t; }
  public void run() {
    try {
      other.join();
    } catch (Exception e) { }
    for (int x = 0; x < 100; x++) {
      try {
        Thread.sleep(5);
      } catch (Exception e) { }
       System.out.print("Stimpy ");
    }
  }
}
```

```
  void start() {
    InnerRun ir = new InnerRun();
    Thread t = new Thread(ir);
    InnerRunTwo irr = new InnerRunTwo(t);
    Thread u = new Thread(irr);
     t.start();
     u.start();
       }
   public static void main (String[] args) {
     ThreadTest tt = new ThreadTest();
      tt.start();   }
}
```

What is the result?

- ○ A Ren Ren Ren... ...Stimpy Stimpy Stimpy
- ○ B Stimpy Stimpy Stimpy... ...Ren Ren Ren
- ○ C A random mixture of Rens and Stimpys
- ◉ D The output cannot be determined.
- ○ E Compilation error.
- ○ F An exception is thrown at runtime.

答案是：A

211. Arrays.asList 方法的用法：

asList

```
public static <T> List<T> asList(T... a)
```

返回一个受指定数组支持的固定大小的列表。（对返回列表的更改会“直接写”到数组。）此方法同 Collection.toArray() 一起，充当了基于数组的 API 与基于 collection 的 API 之间的桥梁。返回的列表是可序列化的，并且实现了 RandomAccess。

此方法还提供了一个创建固定长度的列表的便捷方法，该列表被初始化为包含多个元素：

```
List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");
```

参数：

a - 支持列表的数组。

返回：

指定数组的列表视图。

PriorityQueue 方法

boolean	**add**(E e) 将指定的元素插入此优先级队列。
void	**clear**() 从此优先级队列中移除所有元素。
Comparator<? super E>	**comparator**() 返回用来对此队列中的元素进行排序的比较器；如果此队列根据其元素的自然顺序进行排序，则返回 null。
boolean	**contains**(Object o) 如果此队列包含指定的元素，则返回 true。
Iterator<E>	**iterator**() 返回在此队列中的元素上进行迭代的迭代器。
boolean	**offer**(E e) 将指定的元素插入此优先级队列。
E	**peek**() 获取但不移除此队列的头；如果此队列为空，则返回 null。
E	**poll**() 获取并移除此队列的头，如果此队列为空，则返回 null。
boolean	**remove**(Object o) 从此队列中移除指定元素的单个实例（如果存在）。
int	**size**() 返回此 collection 中的元素数。
Object[]	**toArray**() 返回一个包含此队列所有元素的数组。
<T> T[]	**toArray**(T[] a) 返回一个包含此队列所有元素的数组；返回数组的运行时类型是指定数组的类型。

从类 java.util.AbstractQueue 继承的方法

212.参数传递的进一步理解。值传递

68

Given:

```
1.  public class Test {
2.    public static void main(String [] args) {
3.      Test t = new Test();
4.      t.start();
5.    }
7.    void start() {
8.      String s1 = "one";
9.      String s2 = alter(s1);
10.     System.out.println(s1 + " " + s2);
11.   }
13.   String alter(String s1) {
14.     s1 = s1 + " two";
15.     System.out.print(s1 + " ");
16.     return "three";
17.   }
18. }
```

What is the result?

String 类型在参数中传递时，在调用方法中的是备份。调用方法中进行的操作实际上是在堆中创建了一个新的对象指向了这个备份变量。原来的变量并没有发生变化。

```
    try {
        Date time=sdf.parse("2012-02-12");
        alter2(time);
        List a=new ArrayList();
        a.add("1");
        alter3(a);
        Person p=new Person();
        p.name="old one";
        alter4(p);
        System.out.println("---------------1."+a);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

```
void alter2(Date time){
    time=new Date();
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
    System.out.println("---------------2."+sdf.format(time));
}

void alter3(List list){
    List list2=new ArrayList();
    list2.add("2");
    list=list2;
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
    System.out.println("---------------2."+list);
}

void alter4(Person p){
    Person p1=new Person();
    p1.name="new One";
    p=p1;
    p.name="new One";
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
    System.out.println("---------------2."+p);
}
```

看上面的 DATE 方法。结果是并没有被改变。只要在子类中使用 new 关键字创建了新对象的都不会改变原来的对象。没有使用 new 关键字的则会改变。值传递的理解。

213. 子类和父类强制转换时注意。子类可以被强制转换成父类。

24

Given:

```
class Under extends Mid {
  static String s = " ";
  public static void main(String[] args) {
    Upper u = new Under();
    s = u.go();
    Mid m = (Mid)u;
    System.out.println(s += m.go());
  }
}
class Upper { String go() { return "hi "; } }
class Mid extends Upper { }
```

What is the result?

答案：HI HI

214.对于集合泛型的理解:

59

Given:

```
import java.util.*;
class Apple { }
class Macintosh extends Apple {
  public static void main(String[] munch) {
    List<Apple> a = new ArrayList<Apple>();
    basket(a);
  }
     // insert code here
}
```

Which inserted at // insert code here will compile? (Choose all that apply.)

- [x] A static void basket(List<? super Apple> list) {list.add(new Object()); }
- [ ] B static void basket(List<? super Apple> list) {list.add(new Apple()); }
- [ ] C static void basket(List<Apple> list) {list.add(new Object()); }
- [x] D static void basket(List<Apple> list) {list.add(new Apple()); }
- [x] E static void basket(List<?> list) {list.add(new Apple()); }
- [x] F static void basket(List<?> list) {list.size(); }

答案:B　D　F　Object 不是 Apple 的子类。所以 A 错误。？表示任意的类型，但不能进行添加操作。

215.断言理解：

17

Given:

```
1.  public class Test {
2.    public static int y;
3.    public static int foo(int x) {
4.      System.out.print("foo ");
5.      return y = x;
6.    }
7.    public static int bar(int z) {
8.      System.out.print("bar ");
9.      return y = z;
10.   }
11.   public static void main(String [] args ) {
12.     int t = 2;
13.     assert t < 4 : bar(7);
14.     assert t > 1 : foo(8);
15.     System.out.println("done ");
16.   }
17. }
```

What is the result?

- ○ A done
- ◉ B bar done
- ○ C foo done
- ○ D bar foo done
- ○ E Compilation fails.

答案是:A

216.使用 Arrays.asList 方法时，如果改变返回的集合。则原来的对象也会发生变化。改变原来的对象则返回的集合也会发生变化。

61

Given:

```
import java.util.*;
 class Metric {
   public static void main(String[] args) {
     String[] s = {"inch", "foot", "yard", "rod", "meter"};
     List list = Arrays.asList(s);
     list.set(4, "chain");
     System.out.print(s[4]);
     s[2] = "meter";
     System.out.print(" " + list.get(2));
   }
 }
```

What is the result?

- ○ A meter yard
- ○ B chain yard
- ○ C chain meter
- ○ D meter meter
- ○ E Compilation fails.
- ◉ F An exception is thrown at runtime.

答案：C

217.对于排序集合，不能方法类型不一样的元素。

57

Given:

```
  public static void main(String[] args) {
    TreeSet<Number> tree = new TreeSet<Number>();
    tree.add(108);
    // INSERT HERE
  }
```

Which of the following may be inserted at // INSERT HERE to compile and run without error?

- ☑ A tree.add(3.14);
- ☐ B tree.add("42");
- ☑ C tree.add(Integer.valueOf("-1"));
- ☐ D tree.add(null);
- ☑ E tree.add(new Long(123));
- ☑ F tree.add(0xCAFE);
- ☐ G None of the above.

答案是： C F.

但才逻辑运算中可以把整数和小数进行比较。

```
if(10>11.4){
    System.out.println();
}
```

218. Can only iterate over an array or an instance of java.lang.Iterable

```
    List list2=new ArrayList();
    list2.add("1");
    list2.add("2");
    list2.add("3");
    for (Object obj:reverse(list2)) {

    }
}
public static Iterator reverse(List list){
    Collections.reverse(list);
    return list.iterator();

}
```

218.注意多态时的异常处理：父类没有异常时，子类不能有异常抛出。父类有异常抛出时，子类可以不抛异常。

```
5. class A {
6. void foo() throws Exception { throw new Exception(); }
7. }
8. class SubB2 extends A {
9. void foo() { System.out.println("B "); }
10. }
11. class Tester {
12. public static void main(String[] args) {
13. A a = new SubB2();
14. a.foo();
15. }
16. }
```

A. B

B. B， 後面是 Exception。

C. 編譯會失敗， 因為第 9 行有錯誤。

D. 編譯會失敗， 因為第 14 行有錯誤。

E. 丟出 Exception， 沒有其他輸出。

Answer: D

219.对于 StringBuffer 类，delete、replace、substring 方法的 start 参数都是包含的。而 end 参数都是不包含的。Replace 方法有 3 个参数。都是从 0 开始计算的。

220.

14. 已知下列目錄結構: bigProject |--source| |--Utils.java| |--classes|--

以及下列指令行呼叫: javac -d classes source/Utils.java 假設目前的目錄是 bigProject，則會有什麼結果?

A. 如果編譯成功，則會把 Utils.class 加入來源目錄。

B. 編譯器會傳回無效的旗標錯誤。

C. 如果編譯成功，則會把 Utils.class 加入類別目錄。

D. 如果編譯成功，則會把 Utils.class 加入 bigProject 目錄。

Answer: C

221.注意多变元参数必须要放到最后一个。

```
public void go(int x,String... y){
    System.out.println(y[y.length-1]);
}
```

222.静态内部类的实例化：

```
10. class Line {
11. public static class Point {}
12. }
13.
14. class Triangle {
15. // insert code here
16. }
```

在第 15 行插入哪段程式碼可建立 Line 中定義的 Point 類別實例?

A. Point p = new Point();

B. Line.Point p = new Line.Point();

C. Point 類別無法在第 15 行實例化。

D. Line l = new Line() ; l.Point p = new l.Point();

Answer: B

223.swich 语句容易忽略的地方。Default 不能和 case 一起使用：

```
13          switch (myDog) {
14           case collie:
15           System.out.print("collie ");
16           case default:
17               System.out.print("retriever ");
18           case harrier:
19           System.out.print("harrier ");
20           }
21          }
```

常用单词:

Delimiter：[di'limitə]定界符

Comma：['kɔmə] 逗号；停顿

Ordinal：['ɔ:dinəl] 序数

Irrelevant：[ɪ'relɪv(ə)nt] 不相干的；不切题的

polymorphism[,pɑlɪ'mɔrfɪzm] 多态

priority[praɪ'ɔrəti] n. 优先；优先权；[数] 优先次序；优先考虑的事

verify ['verifai] vt. 核实；查证

eligible ['elidʒəbl] adj. 合格的，合适的；符合条件的；有资格当选的
n. 合格者；适任者；有资格者

reuse [,ri:'ju:z, ,ri:'ju:s, 'ri:ju:s] n. 重新使用，再用
vt. 再使用

correction [kə'rekʃən] n. 改正，修正

consequence ['kɔnsi,kwəns] n. 结果；重要性；推论

reliability [ri,laiə'biləti] n. 可靠性
covariant [kəu'vɛəriənt] adj. 协变的
n. 共变式；协变量
nested ['nestid] adj. 嵌套的，内装的
v. 筑巢；嵌入（nest 的过去分词）
cohesive [kəu'hi:siv] adj. 有结合力的；紧密结合的；有粘着力的