**Java重难点面试（一）**

原创 2016年08月16日 17:07:23

**1 . 以下集合对象中哪几个是线程安全的？(B,C,D )**

A: ArrayList   
B: Vector   
C: Hashtable   
D: Stack

解析：下面是这些线程安全的同步的类：   
vector：就比arraylist多了个同步化机制（线程安全），因为效率较低，现在已经不太建议使用。在web应用中，特别是前台页面，往往效率（页面响应速度）是优先考虑的。   
statck：堆栈类，先进后出   
hashtable：就比hashmap多了个线程安全   
enumeration：枚举，相当于迭代器   
除了这些之外，其他的都是非线程安全的类和接口。

**2 . given the following code,what will be the output?**

class Value{

public int i=15;

}

public class Test{

public static void main(String argv[]){

Test t=new Test( );

t.first( );

}

public void first( ){

int i=5;

Value v=new Value( );

v.i=25;

second(v,i);

System.out.println(v.i);

}

public void second(Value v,int i){

i = 0;

v.i = 20;

Value val = new Value( );

v = val;

System.out.println(v.i+" "+i);

}

结果：15 0 25

这里考查的是Java中的值传递和引用传递的知识点，在Java中基本数据类型和String作为参数时都是值传递，不能够修改传递之前的值，形参不会影响实参，在Java中对象类型作为参数时都是引用传递，只能够修改引用指向的内容，而不能修改引用的指向。

**3 . 下面代码的输出结果是什么？**

public class ZeroTest {

public static void main(String[] args) {

try{

int i = 100 / 0;

System.out.print(i);

}catch(Exception e){

System.out.print(1);

throw new RuntimeException();

}finally{

System.out.print(2);

}

System.out.print(3);

}

}

输出的结果是 12

考查的是finally不管是否抛出异常，都会指向finally里面的语句

**4 . Java类Demo中存在方法func0、func1、func2、func3和func4，请问该方法中，哪些是不合法的定义？( )**

public class Demo{

　　float func0()

　　{

　　　　byte i=1;

　　　　return i;

　　}

　　float func1()

　　{

　　　　int i=1;

　　　　return;

　　}

　　float func2()

　　{

　　　　short i=2;

　　　　return i;

　　}

　　float func3()

　　{

　　　　long i=3;

　　　　return i;

　　}

　　float func4()

　　{

　　　　double i=4;

　　　　return i;

　　}

}

定义错误的是 func1,func4

考查的知识点是自动转型问题，精度高的转型成精度低的需要强制转化申明，精度低的转化成精度高的隐式转化即可。   
低 ———————————————> 高   
byte,short,char-> int -> long -> float -> double   
特殊点：long到float不用强转，因为float表示的范围确实比long表示的范围大，虽然它只占4个字节，而long占8个字节

**5 . 在java中，下列标识符不合法的有（ ）**

A: new   
B: $Usdollars   
C: 1234   
D: car.taxi

答案：A,C,D

考查命名规则：Java标识符由数字，字母和下划线（\_），美元符号（$）组成。在Java中是区分大小写的，而且还要求首位不能是数字。最重要的是，Java关键字 不能当作Java标识符。

**6 . java中将ISO8859-1字符串转成GB2312编码，语句为 ？**

A: new String(String.getBytes(“ISO8859-1”），GB2312)   
B: new String(String.getBytes(“GB2312”），ISO8859-1)   
C: new String(String.getBytes(“ISO8859-1”))   
D: new String(String.getBytes(“GB2312”))

答案 A

**7 . 下列哪个选项是Java调试器？如果编译器返回程序代码的错误，可以用它对程序进行调试。**

A: java.exe   
B: javadoc.exe   
C: jdb.exe   
D: javaprof.exe

答案为C

javac.exe是编译.java文件   
java.exe是执行编译好的.class文件   
javadoc.exe是生成Java说明文档   
jdb.exe是Java调试器   
javaprof.exe是剖析工具

**8 . String与StringBuffer的区别。**

String 字符串常量   
StringBuffer 字符串变量（线程安全）   
StringBuilder 字符串变量（非线程安全）   
String 类代表字符串。Java 程序中的所有字符串字面值（如 “abc” ）都是String类的对象。字符串是常量；它们的值在创建之后不能改变。所以可以共享它们。StringBuffer是字符串缓存区，它的内容可以被修改，长度也可以改变，StringBuffer类是线程安全的，也就是说多个线程可以安全的访问StringBuffer对象。StringBuilder与StringBuffer类似，都是字符串缓冲区，但StringBuilder不是线程安全的，如果你只是在单线程中使用字符串缓冲区，那么StringBuilder的效率会更高些。值得注意的是StringBuilder是在JDK1.5版本中增加的。以前版本的JDK不能使用该类。

**9 . 下列说法错误的有（ ）**

A: 在类方法中可用this来调用本类的类方法   
B: 在类方法中调用本类的类方法时可直接调用   
C: 在类方法中只能调用本类中的类方法   
D: 在类方法中绝对不能调用实例方法

答案为A,C,D

知识点：this是调用对象方法使用的，在类方法中可以通过类名作用域的方式调用Class::fun();   
d，太绝对化了，在类中申请一个类对象或者参数传递一个对象或者指针都可以调用;

**10 . 下面有关java classloader说法错误的是?**

A: Java默认提供的三个ClassLoader是BootStrap ClassLoader，Extension ClassLoader，App ClassLoader   
B: ClassLoader使用的是双亲委托模型来搜索类的   
C: JVM在判定两个class是否相同时，只用判断类名相同即可，和类加载器无关   
D: ClassLoader就是用来动态加载class文件到内存当中用的

答案为C

知识点：   
一个jvm中默认的classloader有Bootstrap ClassLoader、Extension ClassLoader、App ClassLoader，分别各司其职：   
Bootstrap ClassLoader 负责加载java基础类，主要是 %JRE\_HOME/lib/ 目录下的rt.jar、resources.jar、charsets.jar和class等

Extension ClassLoader 负责加载java扩展类，主要是 %JRE\_HOME/lib/ext 目录下的jar和class

App ClassLoader 负责加载当前java应用的classpath中的所有类。

classloader 加载类用的是全盘负责委托机制。 所谓全盘负责，即是当一个classloader加载一个Class的时候，这个Class所依赖的和引用的所有 Class也由这个classloader负责载入，除非是显式的使用另外一个classloader载入。   
所以，当我们自定义的classlo ader加载成功了 com.company.MyClass以后，MyClass里所有依赖的class都由这个classLoader来加载完成。

**11 . 选出不正确的叙述。**

A: 死锁是指因相互竞争资源使得系统中有多个阻塞进程的情况   
B: 若系统中并发运行的进程和资源之间满足互斥条件、占有且申请、不可抢占和环路条件，则可判定系统中发生了死锁   
C: 在对付死锁的策略中，解除死锁通常都是和检测死锁配套使用   
D: 产生死锁的原因可归结为竞争资源和进程推进顺序不当   
E: 在死锁的解决方法中，由于避免死锁采用静态分配资源策略，所以对资源的利用率不高

答案为A,B

知识点：   
死锁是指因相互竞争资源使得系统中有多个阻塞进程相互等待的情况。   
互斥，占有申请，非剥夺，循环等待只是死锁的必要条件

**12 . 在java中重写方法应遵循规则的包括（）**

A: 访问修饰符的限制一定要大于被重写方法的访问修饰符   
B: 可以有不同的访问修饰符   
C: 参数列表必须完全与被重写的方法相同   
D: 必须具有不同的参数列表

答案为B,C

知识点：   
方法的重写（override）两同两小一大原则：   
方法名相同，参数类型相同   
子类返回类型小于等于父类方法返回类型，   
子类抛出异常小于等于父类方法抛出异常，   
子类访问权限大于等于父类方法访问权限。

**13 . 关于ThreadLocal以下说法正确的是**

A: ThreadLocal继承自Thread   
B: ThreadLocal实现了Runnable接口   
C: ThreadLocal重要作用在于多线程间的数据共享   
D: ThreadLocal是采用哈希表的方式来为每个线程都提供一个变量的副本   
E: ThreadLocal保证各个线程间数据安全，每个线程的数据不会被另外线程访问和破坏

答案为D,E

知识点：   
1、ThreadLocal的类声明：   
public class ThreadLocal   
可以看出ThreadLocal并没有继承自Thread，也没有实现Runnable接口。所以AB都不对。   
2、ThreadLocal类为每一个线程都维护了自己独有的变量拷贝。每个线程都拥有了自己独立的一个变量。   
所以ThreadLocal重要作用并不在于多线程间的数据共享，而是数据的独立，C选项错。   
由于每个线程在访问该变量时，读取和修改的，都是自己独有的那一份变量拷贝，不会被其他线程访问，   
变量被彻底封闭在每个访问的线程中。所以E对。   
3、ThreadLocal中定义了一个哈希表用于为每个线程都提供一个变量的副本：

static class ThreadLocalMap {

static class Entry extends WeakReference<ThreadLocal> {

/\*\* The value associated with this ThreadLocal. \*/

Object value;

Entry(ThreadLocal k, Object v) {

super(k);

value = v;

}

}

/\*\*

\* The table, resized as necessary.

\* table.length MUST always be a power of two.

\*/

private Entry[] table;

}

所以D对。

**14 .**

public class Test

{

public int x;

public static void main(String args[])

{

System. out. println("Value is" + x);

}

}

对于上面这段代码，以下说法正确的是：

A: 程序会打出 “Value is 0”   
B: 程序会抛出 NullPointerException   
C: 非静态变量不能够被静态方法引用   
D: 编译器会抛出 “possible reference before assignment”的错误

答案为C

知识点：假如变量没有赋初始值，默认初始值为0，静态方法不能够使用静态变量。

**15 .以下方法，哪个不是对add方法的重载?**

public class Test   
{   
public void add( int x,int y,int z){}   
}

A: public int add(int x,int y,float z){return 0;}   
B: public int add(int x,int y,int z){return 0;}   
C: public void add(int x,int y){}   
D: 以上都不是

答案为B

知识点：   
区分不同重载函数的方法：**注意点：返回值不同不能作为区分重载函数的标准**

一.方法名一定要相同。   
二.方法的参数表必须不同，包括参数的类型或个数，以此区分不同的方法体。   
1.如果参数个数不同，就不管它的参数类型了！   
2.如果参数个数相同，那么参数的类型或者参数的顺序必须不同。   
三.方法的返回类型、修饰符可以相同，也可不同。

**16 . class Foo {**

final int i;

int j;

public void doSomething() {

System.out.println(++j + i);

}

}   
的输出是？

A: 0   
B: 1   
C: 2   
D: 不能执行，因为编译有错

答案为 D

知识点：final 修改时的i必须要赋初始值。

**17 . java用（）机制实现了进程之间的异步执行**

A: 监视器   
B: 虚拟机   
C: 多个CPU   
D: 异步调用

答案为 A

知识点：首先jvm中没有进程的概念，但是jvm中的线程映射为操作系统中的进程，对应关系为1：1。那这道题的问的就是jvm中线程如何异步执行 。 在jvm中 是使用监视器锁来实现不同线程的异步执行， 在语法的表现就是synchronized 。

**18 . 关于volatile关键字，下列描述不正确的是？**

A: 用volatile修饰的变量，每次更新对其他线程都是立即可见的。   
B: 对volatile变量的操作是原子性的。   
C: 对volatile变量的操作不会造成阻塞。   
D: 不依赖其他锁机制，多线程环境下的计数器可用volatile实现。

答案为：B,

知识点：   
一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，那么就具备了两层语义：   
1）保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。   
2）禁止进行指令重排序。   
volatile只提供了保证访问该变量时，每次都是从内存中读取最新值，并不会使用寄存器缓存该值——每次都会从内存中读取。   
而对该变量的修改，volatile并不提供原子性的保证。   
由于及时更新，很可能导致另一线程访问最新变量值，无法跳出循环的情况   
多线程下计数器必须使用锁保护。

**19 . What will be printed when you execute the following code?**

class C {   
C() {   
System.out.print(“C”);   
}   
}

class A {   
C c = new C();

A() {

this("A");

System.out.print("A");

}

A(String s) {

System.out.print(s);

}

}

class Test extends A {   
Test() {   
super(“B”);   
System.out.print(“B”);   
}

public static void main(String[] args) {

new Test();

}

}

答案为 CBB

知识点：考查类的执行顺序

父类的静态变量-父类的静态块-子类的静态变量-子类的静态块-父类的普通变量，代码块，构造函数-子类的普通变量，代码块，构造函数

**20 . Which statement declares a variable a which is suitable for referring to an array of 50 string objects?**

A: char a[][];   
B: String a[];   
C: String[]a;   
D: Object a[50];   
E: String a[50];   
F: Object a[];

答案为 B,C,F

知识点：

在java 中，声明一个数组时，不能直接限定数组长度，只有在创建实例化对象时，才能对给定数组长度.。   
如下，1，2,3可以通过编译，4，5不行。而String是Object的子类，所以上述BCF均可定义一个存放50个String类型对象的数组。   
1. String a[]=new String[50];   
2. String b[];   
3. char c[];   
4. String d[50];   
5. char e[50];

**21 . 下面哪个不对？**

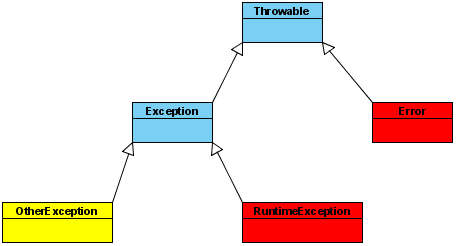
A: RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

B: A method is not required to declare in its throws clause any subclasses of RuntimeExeption that might be thrown during the execution of the method but not caught

C: An RuntimeException is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.

D: NullPointerException is one kind of RuntimeException

答案为 C

知识点：   


运行时错误一般是逻辑错误，数据越界，空指针等。

运行时异常： 都是RuntimeException类及其子类异常，如NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。运行时异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过。

非运行时异常 （编译异常）： 是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常，一般情况下不自定义检查异常。

**22 . 下面代码运行结果是（）**

public class Test{

public int add(int a,int b){

try {

return a+b;

}

catch (Exception e) {

System.out.println("catch语句块");

}

finally{

System.out.println("finally语句块");

}

return 0;

}

public static void main(String argv[]){

Test test =new Test();

System.out.println("和是："+test.add(9, 34));

}

}

A: catch语句块   
和是：43

B: 编译异常

C: finally语句块   
和是：43

D: 和是：43   
finally语句块

答案为 C

知识点：不管trycatch是否抛出异常，finally必定会在return 退出函数前，调用。

**23 . 以下哪个不是Collection的子接口？**

A: List   
B: Set   
C: SortedSet   
D: Map

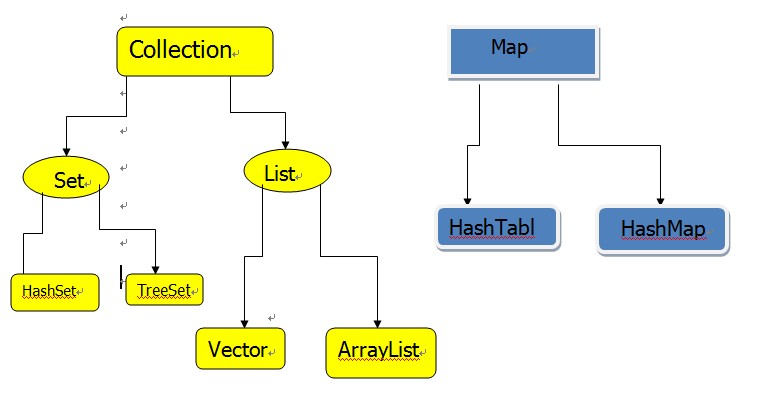
答案为D

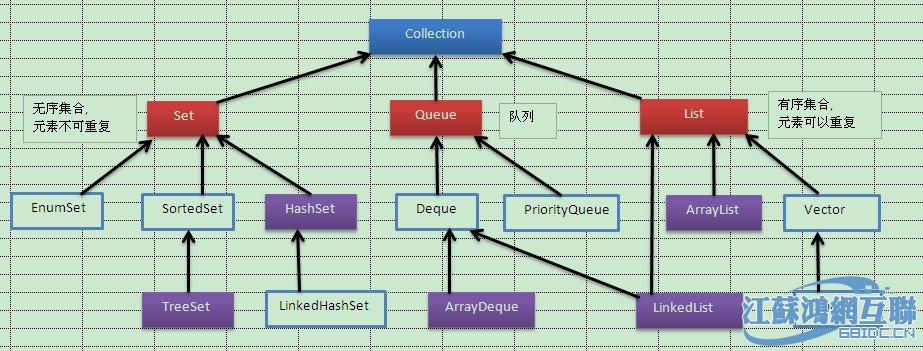
知识点：考查集合的接口分类

list里面可以存放相同的元素，并且有序

set里面不可以存放相同的元素，通过hashcode和equal来区别

vector是线程安全的，hashtable,stack也是线程安全的





**24 . 下面有关 JAVA 异常类的描述,说法正确的有()**

A: 异常的继承结构:基类为 Throwable,Error 和 Exception 继承 Throwable,RuntimeException 和 IOException 等继承 Exception

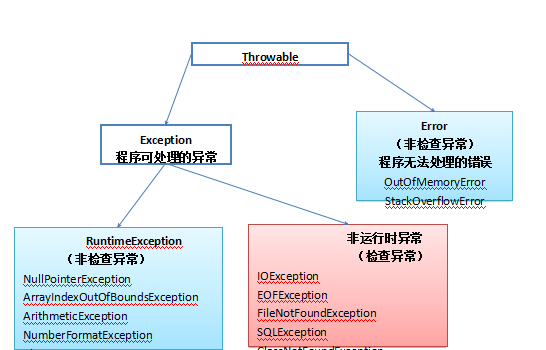
B: 非 RuntimeException 一般是外部错误,其必须被 try{}catch 语句块所捕获

C: Error 类体系描述了 Java 运行系统中的内部错误以及资源耗尽的情形,Error 不需要捕捉

D: RuntimeException 体系包括错误的类型转换、数组越界访问和试图访问空指针等等,必须 被 try{}catch 语句块所捕获

答案为 A,C

知识点：



都是Throwable的子类：   
1.Exception（异常） :是程序本身可以处理的异常。   
2.Error（错误）: 是程序无法处理的错误。这些错误表示故障发生于虚拟机自身、或者发生在虚拟机试图执行应用时，一般不需要程序处理。   
3.检查异常（编译器要求必须处置的异常） ： 除了Error，RuntimeException及其子类以外，其他的Exception类及其子类都属于可查异常。这种异常的特点是Java编译器会检查它，也就是说，当程序中可能出现这类异常，要么用try-catch语句捕获它，要么用throws子句声明抛出它，否则编译不会通过。   
4.非检查异常(编译器不要求处置的异常): 包括运行时异常（RuntimeException与其子类）和错误（Error）。

**25 . 如下代码，执行test()函数后，屏幕打印结果为（）**

public class Test2

{

public void add(Byte b)

{

b = b++;

}

public void test()

{

Byte a = 127;

Byte b = 127;

add(++a);

System.out.print(a + " ");

add(b);

System.out.print(b + "");

}

}

A: 127 127   
B: 128 127   
C: 129 128   
D: 以上都不对

答案为D

正确的答案为，-128,127

知识点：byte是八位，最高位为符号位，数据范围为-128~127，当127++是数值溢出，变为-128，在Java中使用补码表示负数，**补码就是将正数的所有二进制取反，最后加1**