

以下代码执行的结果显示是多少 ( ) ?

```
public class Demo {  
    public static void main(String args[]) {  
        int count = 0;  
        int num = 0;  
        for (int i = 0; i <= 100; i++) {  
            num = num + i;  
            count = count++;  
        }  
        System.out.println("num * count = " + (num * count));  
    }  
}
```

正确答案: B 你的答案: 空 (错误)

num \* count = 505000

num \* count = 0

运行时错误

num \* count = 5050

在 TCP/IP 建立连接过程中, 客户端和服务端的状态转移说法错误的是 ( ) ?

正确答案: A 你的答案: 空 (错误)

经历 TIME\_WAIT 状态

经历 SYN\_SENT 状态

经历 ESTABLISHED 状态

经历 SYN\_RECV 状态

服务器接受到客户端的 ack 包后将从半连接队列删除

服务器在收到 syn 包时将加入半连接队列

假设栈的输入序列是 7,6,2,1,4, 则以下不可能是其出栈序列 ( ) ?

正确答案: D 你的答案: 空 (错误)

6,7,4,1,2

其它都不是

7,6,2,1,4

6,7,4,2,1

下面哪个行为被打断不会导致 InterruptedException: ( ) ?

正确答案: E 你的答案: 空 (错误)

Thread.join

Thread.sleep

Object.wait

CyclicBarrier.await

Thread.suspend

以下代码执行的结果显示是多少 ( ) ?

```

public class Demo {
    public static void main(String[] args) {
        Integer i1 = 128;
        Integer i2 = 128;
        System.out.print((i1 == i2) + ",");
        String i3 = "100";
        String i4 = "1" + new String("00");
        System.out.print((i3 == i4) + ",");
        Integer i5 = 100;
        Integer i6 = 100;
        System.out.println((i5 == i6));
    }
}

```

正确答案: D 你的答案: 空 (错误)

true,false,true

false,true,false

true,true,false

false,false,true

以下哪个式子有可能在某个进制下成立 ( ) ?

正确答案: A 你的答案: 空 (错误)

13\*14=204

12\*34=568

14\*14=140

1+1=3

某文件占 10 个磁盘块，现要把该文件磁盘块逐个读入主存缓冲区，并送用户区进行分析。假设一个缓冲区与一个磁盘块大小相同，把一个磁盘块读入缓冲区时间为  $80\mu\text{s}$ ，将缓冲区的数据传送到用户区的时间是  $60\mu\text{s}$ ，CPU 对一块数据进行分析的时间为  $40\mu\text{s}$ 。在单缓冲区和双缓冲区结构下，读入并分析完该文件的时间分别是 ( )。

正确答案: C 你的答案: 空 (错误)

1260 $\mu\text{s}$ , 840 $\mu\text{s}$

1440 $\mu\text{s}$ , 840 $\mu\text{s}$

1440 $\mu\text{s}$ , 900 $\mu\text{s}$

1260 $\mu\text{s}$ , 900 $\mu\text{s}$

33 人围成一圈，从 1 至 N 开始顺时针一直递增报数；报 N 者退出，下一位从 1 开始重新报数。当 N 为 2 时，最后留下者是第几人。假设最先报数的人编号是 1，其他人编号按顺时针方向递增。( )

正确答案: C 你的答案: 空 (错误)

7

9

3

5

以下代码执行的结果显示是多少 ( ) ?

```
1    public class Demo {
2        class Super {
3
4            int flag = 1;
5
6            Super() {
7                test();
8            }
9
10           void test() {
11               System.out.println("Super.test() flag=" + flag);
12           }
13       }
14       class Sub extends Super {
15
16           Sub(int i) {
17               flag = i;
18               System.out.println("Sub.Sub()flag=" + flag);
19           }
20           void test() {
21               System.out.println("Sub.test()flag=" + flag);
22           }
23       }
24       public static void main(String[] args) {
25           new Demo().new Sub(5);
26       }
27   }
```

正确答案: A 你的答案: 空 (错误)

Sub.test() flag=1

Sub.Sub() flag=5

Sub.Sub() flag=5

Sub.test() flag=5

Sub.test() flag=0

Sub.Sub() flag=5

Super.test() flag=1

Sub.Sub() flag=5

以下描述错误的一项是 ( ) ?

正确答案: C 你的答案: 空 (错误)

程序计数器是一个比较小的内存区域, 用于指示当前线程所执行的字节码执行 到了第几行, 是线程隔离的

原则上讲, 所有的对象都是在堆区上分配内存, 是线程之间共享的

方法区用于存储 JVM 加载的类信息、常量、静态变量，即使编译器编译后的代码等数据，是线程隔离的

Java 方法执行内存模型，用于存储局部变量，操作数栈，动态链接，方法出口等信息，是线程隔离的

```
1 public interface IService {  
2     String NAME="default";  
3 }
```

与上面等价表示是哪一项 ( ) ?

正确答案: D 你的答案: 空 (错误)

```
public String NAME="default";  
public static String NAME="default";  
private String NAME="default";  
public static final String NAME="default";
```

socket 编程中，以下哪个 socket 的操作是不属于服务端操作的 ( ) ?

正确答案: C 你的答案: 空 (错误)

```
accept  
listen  
connect  
close
```

关于 JAVA 堆，下面说法错误的是 ( ) ?

正确答案: C 你的答案: 空 (错误)

所有类的实例和数组都是在堆上分配内存的  
堆内存由存活和死亡的对象，空闲碎片区组成  
数组是分配在栈中的  
对象所占的堆内存是由自动内存管理系统回收

下面论述正确的是 ( ) ?

正确答案: D 你的答案: 空 (错误)

如果两个对象的 hashCode 相同，那么它们作为同一个 HashMap 的 key 时，必然返回同样的值

如果 a,b 的 hashCode 相同，那么 a.equals(b)必须返回 true

对于一个类，其所有对象的 hashCode 必须不同

如果 a.equals(b)返回 true，那么 a,b 两个对象的 hashCode 必须相同

关于 ASCII 码和 ANSI 码，以下说法不正确的是 ( ) ?

正确答案: D 你的答案: 空 (错误)

标准 ASCII 只使用 7 个 bit

在简体中文的 Windows 系统中，ANSI 就是 GB2312

ASCII 码是 ANSI 码的子集

ASCII 码都是可打印字符

TCP 协议头中不包含哪些字段 ( ) ?

正确答案: B 你的答案: 空 (错误)

校验和

源 IP 地址和目的 IP 地址

序列号和确认号

源端口和目的端口

具有八个结点的二叉树共有多少种 ( ) ?

正确答案: D 你的答案: 空 (错误)

8

256

960

1430

执行下列代码后, 哪些结论是正确的 ( ) ?

`String[] s=new String[10];`

正确答案: B 你的答案: 空 (错误)

`s[0]`为未定义

`s[9]`为 null

`s.length` 为 0

`s[10]`为 " "

在 Java 中, 关于 `HashMap` 类的描述, 以下错误的是 ( ) ?

正确答案: A 你的答案: 空 (错误)

`HashMap` 能够保证其中元素的顺序

`HashMap` 允许将 null 用作值

`HashMap` 允许将 null 用作键

`HashMap` 使用键/值得形式保存数据

关于依赖注入, 下列选项中说法错误的是 ( ) ?

正确答案: C 你的答案: 空 (错误)

依赖注入能够独立开发各组件, 然后根据组件间关系进行组装

依赖注入提供使用接口编程

依赖注入使组件之间相互依赖, 相互制约

依赖注入指对象在使用时动态注入

以下哪个不属于 JVM 堆内存中的区域 ( ) ?

正确答案: B 你的答案: 空 (错误)

survivor 区

常量池

eden 区

old 区

下列关于管道 (Pipe) 通信的叙述中, 正确的是 ( ) ?

正确答案: A 你的答案: 空 (错误)

进程对管道进行读操作和写操作都可能被阻塞

一个管道只能有一个进程或一个写进程对其操作

一个管道可实现双向数据传输

管道的容量仅受磁盘容量大小限制

在一个基于分布式的游戏服务器系统中，不同的服务器之间，哪种通信方式是不可行的（）？

正确答案: A 你的答案: 空 (错误)

管道

消息队列

高速缓存数据库

套接字

杀人游戏，6 个人互相投票，有一个人被其他 5 个人一起投死的概率是多少（）？

假设每个人都不会投自己，投其他每个人是等概率的。

正确答案: D 你的答案: 空 (错误)

6/1296

5/1296

5/3125

6/3125

关于 JDBC PreparedStatement，下面说法错误的是（）？

正确答案: C 你的答案: 空 (错误)

可以用来进行动态查询

通过预编译和缓存机制提升了执行的效率

不能直接用它来执行 in 条件语句，但可以动态生成 PreparedStatement，一样

能享受 PreparedStatement 缓冲带来的好处

有助于防止 SQL 注入，因为它会自动对特殊字符转义

transient 变量和下面哪一项有关（）？

正确答案: A 你的答案: 空 (错误)

Serializable

Cloneable

Runnable

Throwable

Comparable

以下代码执行的结果显示是多少（）？

```
public class Demo{  
    public static void main(String[] args){  
        System.out.print(getNumber(0));  
        System.out.print(getNumber(1));  
        System.out.print(getNumber(2));  
        System.out.print(getNumber(4));  
    }  
}
```

```
public static int getNumber(int num){  
    try{  
        int result = 2 / num;  
        return result;  
    }catch (Exception exception){  
        return 0;  
    }  
}
```

```

        }finally{
            if(num == 0){
                return -1;
            }
            if(num == 1){
                return 1;
            }
        }
    }
}

```

正确答案: B 你的答案: 空 (错误)

0110

-1110

0211

-1211

以下代码执行的结果是多少 ( ) ?

```

1      public class Demo {
2          public static void main(String[] args) {
3              Collection<?>[] collections =
4              {new HashSet<String>(), new ArrayList<String>(), new HashMap<String, String>().values()};
5              Super subToSuper = new Sub();
6              for(Collection<?> collection: collections) {
7                  System.out.println(subToSuper.getType(collection));
8              }
9          }
10         abstract static class Super {
11             public static String getType(Collection<?> collection) {
12                 return "Super:collection";
13             }
14             public static String getType(List<?> list) {
15                 return "Super:list";
16             }
17             public String getType(ArrayList<?> list) {
18                 return "Super:arrayList";
19             }
20             public static String getType(Set<?> set) {
21                 return "Super:set";
22             }
23             public String getType(HashSet<?> set) {
24                 return "Super:hashSet";
25             }
26         }
27         static class Sub extends Super {
28             public static String getType(Collection<?> collection) {

```

```
29         return "Sub";}
30     }
31 }
```

正确答案: C 你的答案: 空 (错误)

Sub:collection

Sub:collection

Sub:collection

Sub:hashSet

Sub:arrayList

Sub:collection

Super:collection

Super:collection

Super:collection

Super:hashSet

Super:arrayList

Super:collection

正则表达式中，表示匹配非数字字符的字符是 ( ) ?

正确答案: D 你的答案: 空 (错误)

\b

\d

\B

\D

在一个 IP 数据包到达目的地之前，可能发生很多的情况，以下哪个说法是正确的 ( ) ?

正确答案: D 你的答案: 空 (错误)

不能成为碎片或者重组

不能成为碎片，但是会重组

可能成为碎片或者重组

可能成为碎片，但是不会重组

下列说法正确的是 ( ) ?

正确答案: A B 你的答案: 空 (错误)

对于局部内部类，只有在方法的局部变量被标记为 final 或局部变量是 effectively final 的，内部类才能使用它们

成员内部类位于外部类内部，可以直接调用外部类的所有方法（静态方法和非静态方法）

由于匿名内部类只能用在方法内部，所以匿名内部类的用法与局部内部类是一致的

静态内部类可以直接访问外部类的非静态成员

下面几个关于 Java 里 queue 的说法哪些是正确的 ( ) ?

正确答案: A C 你的答案: 空 (错误)

LinkedBlockingQueue 是一个可选有界队列，不允许 null 值

PriorityQueue, LinkedBlockingQueue 都是线程不安全的

PriorityQueue 是一个无界队列，不允许 null 值，入队和出队的时间复杂度是  $O(\log(n))$



PriorityQueue, ConcurrentLinkedQueue 都遵循 FIFO 原则

下面的对象创建方法中哪些会调用构造方法 () ?

正确答案: A C 你的答案: 空 (错误)

new 语句创建对象

调用 Java.io.ObjectInputStream 的 readObject 方法

java 反射机制使用 java.lang.Class 或 java.lang.reflect.Constructor 的 newInstance()方法

调用对象的 clone()方法

以下哪些方法可以取到 http 请求中的 cookie 值 () ?

正确答案: B D 你的答案: 空 (错误)

request.getAttribute

request.getHeader

request.getParameter

request.getCookies

mysql 数据库, game\_order 表表结构如下, 下面哪些 sql 能使用到索引 () ?

```
CREATE TABLE `game_order` (  
  `id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `plat_id` int(11) NOT NULL,  
  `plat_game_id` int(11) NOT NULL,  
  `plat_uid` varchar(100) NOT NULL,  
  `plat_uname` varchar(100) NOT NULL,  
  `plat_order_id` varchar(100) NOT NULL,  
  `game_order_id` varchar(100) NOT NULL DEFAULT "",  
  `server_id` int(11) NOT NULL DEFAULT '0',  
  `role_id` varchar(100) NOT NULL DEFAULT "",  
  `role_name` varchar(100) NOT NULL DEFAULT "",  
  `amount` int(11) NOT NULL,  
  `pay_time` datetime NOT NULL,  
  `create_time` datetime NOT NULL,  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `ukey` (`plat_order_id`, `plat_game_id`, `plat_id`) USING BTREE,  
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

正确答案: B C D E 你的答案: 空 (错误)

select \* from game\_order where plat\_game\_id=5 and plat\_id=134

select \* from game\_order where plat\_id=134 and

plat\_game\_id=5 and plat\_order\_id='100'

select \* from game\_order where plat\_order\_id='100'

select \* from game\_order where plat\_game\_id=5 and

plat\_order\_id='100' and plat\_id=134

select \* from game\_order where plat\_game\_id=5 and plat\_order\_id='100'

关于 OutOfMemoryError, 下面说法正确的是 () ?

正确答案: A B C 你的答案: 空 (错误)

java.lang.OutOfMemoryError: PermGen space 增加-XX:MaxPermSize 这个参数的值的话, 这个问题通常会得到解决。

java.lang.OutOfMemoryError: Requested array size exceeds VM limit 当你正准备创建一个超过虚拟机允许的大小的数组时, 这条错误将会出现

java.lang.OutOfMemoryError: Java heap space 一般情况下解决这个问题最快的方法就是通过 -Xmx 参数来增加堆的大小

java.lang.OutOfMemoryError: nativeGetNewTLA 这个异常只有在 jRockit 虚拟机时才会碰到  
在 Java 线程状态转换时，下列转换不可能发生的有（）？

正确答案: A C 你的答案: 空 (错误)

初始态->运行态

就绪态->运行态

阻塞态->运行态

运行态->就绪态

如果 Child extends Parent，那么正确的有（）？

正确答案: B C D 你的答案: 空 (错误)

如果 Child 是 class，且只有一个有参数的构造函数，那么必然会调用 Parent 中相同参数的构造函数

如果 Child 是 interface，那么 Parent 必然是 interface

如果 Child 是 interface，那么 Child 可以同时 extends Parent1，Parent2 等多个 interface

如果 Child 是 class，并且没有显示声明任何构造函数，那么此时仍然会调用 Parent 的构造函数

下列说法正确的是（）？

正确答案: B D 你的答案: 空 (错误)

我们直接调用 Thread 对象的 run 方法会报异常，所以我们应该使用 start 方法来开启一个线程

一个进程是一个独立的运行环境，可以被看做一个程序或者一个应用。而线程是在进程中执行的一个任务。Java 运行环境是一个包含了不同的类和程序的单一进程。线程可以被称为轻量级进程。线程需要较少的资源来创建和驻留在进程中，并且可以共享进程中的资源

synchronized 可以解决可见性问题，volatile 可以解决原子性问题

ThreadLocal 用于创建线程的本地变量，该变量是线程之间不共享的

下面有关 java classloader 说法正确的是（）？

正确答案: A C D 你的答案: 空 (错误)

ClassLoader 就是用来动态加载 class 文件到内存当中用的

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

ClassLoader 使用的是双亲委托模型来搜索类的

Java 默认提供的三个 ClassLoader 是 Bootstrap ClassLoader，Extension ClassLoader，App ClassLoader

以上都不正确

**编程题】**

方块消除游戏

题目描述：如下图，有 10\*10 个不同颜色的方块，每个方块可能是红、绿、蓝、黄、紫 5 种颜色之一。当点击其中某一个方块时，如果它有相邻的同颜色方块，则将所有与此方块连续同颜色相邻的方块消除；剩下的方块中，如果下方有空位则向下移动，如果左侧整列都为空位则向左移动。



输入：

输入数据有多组，每组占一行，包括一个或多个正整数，取值范围为 1~100。每个数代表一次点击，数值为点击的方块编号。

上图中的方块初始值定义已为你写好，可以直接粘贴使用：

```
const int RED = 0, GREEN = 1, BLUE = 2, YELLOW = 3, PURPLE = 4;
int p[10][10] = {
    {RED,RED,BLUE,BLUE,GREEN,YELLOW,BLUE,YELLOW,RED,PURPLE},
    {GREEN,GREEN,GREEN,BLUE,RED,PURPLE,RED,YELLOW,YELLOW,BLUE},
    {BLUE,RED,RED,YELLOW,YELLOW,PURPLE,BLUE,GREEN,GREEN,BLUE},
    {YELLOW,RED,BLUE,YELLOW,BLUE,RED,PURPLE,GREEN,GREEN,RED},
    {YELLOW,RED,BLUE,BLUE,PURPLE,GREEN,PURPLE,RED,YELLOW,BLUE},
    {PURPLE,YELLOW,RED,RED,YELLOW,RED,PURPLE,YELLOW,RED,RED},
    {YELLOW,YELLOW,GREEN,PURPLE,GREEN,RED,BLUE,YELLOW,BLUE,GREEN},
    {RED,YELLOW,BLUE,BLUE,YELLOW,GREEN,PURPLE,RED,BLUE,GREEN},
    {GREEN,GREEN,YELLOW,YELLOW,RED,RED,PURPLE,BLUE,BLUE,GREEN},
    {PURPLE,BLUE,RED,RED,PURPLE,YELLOW,BLUE,RED,RED,GREEN}};
```

输出：

对于每个测试实例，要求输出连续各次点击全部完成之后，红、绿、蓝、黄、紫色方块的数量;每个测试实例的输出占一行。

样例输入：

6

6 1

样例输出：

26 18 22 21 13

24 18 22 21 13

```
package xiaoxiaole;
```

```
import java.lang.reflect.Array;
```

```
import java.util.Arrays;
```

```
public class Xiaoxiaole {
```

```
    public static void main(String[] args) {
```

```
        int RED = 0, GREEN = 1, BLUE = 2, YELLOW = 3, PURPLE = 4;
```

```
        int[][] p = new int[][]{
```

```
            {RED,RED,BLUE,BLUE,GREEN,YELLOW,BLUE,YELLOW,RED,PURPLE},
```

```
            {GREEN,GREEN,GREEN,BLUE,RED,PURPLE,RED,YELLOW,YELLOW,BLUE},
```

```
            {BLUE,RED,RED,YELLOW,YELLOW,PURPLE,BLUE,GREEN,GREEN,BLUE},
```

```
            {YELLOW,RED,BLUE,YELLOW,BLUE,RED,PURPLE,GREEN,GREEN,RED},
```

```
            {YELLOW,RED,BLUE,BLUE,PURPLE,GREEN,PURPLE,RED,YELLOW,BLUE},
```

```
            {PURPLE,YELLOW,RED,RED,YELLOW,RED,PURPLE,YELLOW,RED,RED},
```

```
            {YELLOW,YELLOW,GREEN,PURPLE,GREEN,RED,BLUE,YELLOW,BLUE,GREEN},
```

```
            {RED,YELLOW,BLUE,BLUE,YELLOW,GREEN,PURPLE,RED,BLUE,GREEN},
```

```
            {GREEN,GREEN,YELLOW,YELLOW,RED,RED,PURPLE,BLUE,BLUE,GREEN},
```

```
            {PURPLE,BLUE,RED,RED,PURPLE,YELLOW,BLUE,RED,RED,GREEN}};
```

```
        operate(p, 6);
```

```
        operate(p, 1);
```

```
    }
```

```
public static void operate(int p[], int position) {

    int[] count = new int[5]; //count[0] for number of red...count[4] for number of
purple

    for(int i=0; i<p.length; i++) {

        for(int j=0;j<p[i].length;j++) {

            if(p[i][j] == 0) {

                count[0]++;

            }else if (p[i][j] == 1) {

                count[1]++;

            }else if (p[i][j] == 2) {

                count[2]++;

            }else if (p[i][j] == 3) {

                count[3]++;

            }else if (p[i][j] == 4) {

                count[4]++;

            }

        }

    }

}
```

```
//get the click position
```

```
int i = (position-1)/10;
```

```
int j = (position-1)%10;
```

```

//up, down, left, right to compare

int count_de_num = 1; // the number to decrease

if(i!= 0) {

    if(p[i-1][j] == p[i][j]) {

        count_de_num++;

    }

}

if(i!= 9) {

    if(p[i+1][j] == p[i][j]) {

        count_de_num++;

    }

}

if(j!= 0) {

    if(p[i][j-1] == p[i][j]) {

        count_de_num++;

    }

}

if(j!= 9) {

    if(p[i][j+1] == p[i][j]) {

        count_de_num++;

    }

}

```

```
}
```

```
if(count_de_num != 1) {
```

```
    if(p[i][j] == 0) {
```

```
        count[0] = count[0] - count_de_num;
```

```
    }else if (p[i][j] == 1) {
```

```
        count[1] = count[1] - count_de_num;
```

```
    }else if (p[i][j] == 2) {
```

```
        count[2] = count[2] - count_de_num;
```

```
    }else if (p[i][j] == 3) {
```

```
        count[3] = count[3] - count_de_num;
```

```
    }else if (p[i][j] == 4) {
```

```
        count[4] = count[4] - count_de_num;
```

```
    }
```

```
}
```

```
System.out.println(Arrays.toString(count));
```

```
}
```

```
}
```