1、JAVA 中能创建 volatile 数组吗？volatile 能使得一个非原子操作变成原子操作吗？

回答： 能，Java 中可以创建 volatile 类型数组，不过只是一个指向数组的引用，而不是整个数组。

Java 中读取 long 类型变量不是原子的，需要分成两步，如果一个线程正在修改该 long 变量的值，另一个线程可能只能看到该值的一半（前 32 位）。但是 volatile 型的 long 或 double 变量的读写是原子。

2、10 个线程和 2 个线程的同步代码，哪个更容易写？

回答: 从写代码的角度来说，两者的复杂度是相同的，因为同步代码与线程数量是相互独立的。但是同步策略的选择依赖于线程的数量，因为越多的线程意味着更大的竞争，所以你需要利用同步技术，如锁分离，这要求更复杂的代码和专业知识。

3、什么是线程局部变量？

回答： 对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而 ThreadLocal 采用了“以空间换时间”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

4、我们自己写一个容器类，然后使用 for-each 循环吗？

回答：可以，你可以写一个自己的容器类。如果你想使用 Java 中增强的循环来遍历，你只需要实现 Iterable 接口。如果你实现 Collection 接口，默认就具有该属性。

5、说出 5 条 IO 的最佳实践？

回答：

1.使用有缓冲区的 IO 类，而不要单独读取字节或字符。

2.使用 NIO 和 NIO2

3.在 finally 块中关闭流，或者使用 try-with-resource 语句。

4.使用内存映射文件获取更快的 IO。

5.使用非阻塞式而不要使用阻塞式的 IO

6、说出至少 5 点在 Java 中使用线程的最佳实践？

回答：1.对线程命名

2.将线程和任务分离，使用线程池执行器来执行 Runnable 或 Callable。

3.使用线程池

4.如果可以，更偏向于使用 volatile 而不是 synchronized。

5.优先使用并发集合，而不是对集合进行同步。并发集合提供更好的可扩展性。

7、我能在不进行强制转换的情况下将一个 double 值赋值给 long 类

型的变量吗？

回答：不行，因为 double 类型的范围比 long 类型更广，所以必须要进行强制转换。

8、我们能在 Switch 中使用 String 吗？

回答：在 jdk 7 之前，switch 只能支持 byte、short、char、int 这几个基本数据类型和其对应的封装类型。switch 后面的括号里面只能放 int 类型的值，但由于 byte，short，char 类型，它们会 自动 转换为 int 类型（精精度小的向大的转化），所以它们也支持。

jdk1.7 后，整形，枚举类型，boolean，字符串都可以。

jdk1.7 并没有新的指令来处理 switch string,而是通过调用 switch 中 string.hashCode,将 string 转换为 int 从而进行判断。

9、poll()方法和 remove()方法的区别？

回答： poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

10、LinkedList 和 ArrayList 的区别？

回答：

1.ArrayList 和 LinkedList 可想从名字分析，它们一个是 Array(动态数组)的数据结构，一个是 Link(链表)的数据结构，此外，它们两

个都是前者是数组队列，相当于动态数组；后者为双向链表结构，也可当作堆栈、队列、双端队列对 List 接口的实现。

2.当随机访问 List 时（get 和 set 操作），ArrayList 比 LinkedList 的效率更高，因为 LinkedList 是线性的数据存储方式，所以需要移动指针从前往后依次查找。

3.当对数据进行增加和删除的操作时(add 和 remove 操作)，LinkedList 比 ArrayList 的效率更高，因为 ArrayList 是数组，所以在其中进行增删操作时，会对操作点之后所有数据的下标索引造成影响，需要进行数据的移动。

4.从利用效率来看，ArrayList 自由性较低，因为它需要手动的设置固定大小的容量，但是它的使用比较方便，只需要创建，然后添加数据，通过调用下标进行使用；而 LinkedList 自由性较高，能够动态的随数据量的变化而变化，但是它不便于使用。

5.ArrayList 主要控件开销在于需要在 lList 列表预留一定空间；而 LinkList 主要控件开销在于需要存储结点信息以及结点指针信息。

11、hashmap 的扩容问题 new hashmap(19)它的长度是多少？

回答：初始长度是 19，当达到默认加载因子的时候会进行扩容

12、hashtable 为什么是线程安全的？

回答：Hashtable 是线程安全的，其实现方式是在对应的方法上加上 synchronized 关键字，效率不高，不建议使用。目前，如果要使用

线程安全的哈希表的话，推荐使用 ConcurrentHashMap。

13、java 异常处理怎么做？

回答：

1.对代码块用 try..catch 进行异常捕获处理；

2.在 该代码的方法体外用 throws 进行抛出声明，告知此方法的调用者这段代码可能会出现这些异常，你需要谨慎处理。此时有两种情况：1）如果声明抛出的异常是非运行时异常，此方法的调用者必须显示地用 try..catch 块进行捕获或者继续向上层抛出异常；2）如果声明抛出的异常是运行时异常，此方法的调用者可以选择地进行异常捕获处理。

3.在代码块用 throw 手动抛出一个异常对象，此时也有两种情况，跟 2 中的类似：1）如果抛出的异常对象是非运行时异常，此方法的调用者必须显示地用 try..catch 块进行捕获或者继续向上层抛出异常；2）如果抛出的异常对象是运行时异常，此方法的调用者可以选择地进行异常捕获处理。

14、异常处理的作用是什么？

回答： 因为你不可能保证程序不出错，所以使用异常处理来防止程序出错的时候你无从下手的局面，对于调试程序和项目实际开发都是有用的

15、jvm 内存的分配？

回答：1.程序计数器：线程私有，当前线程执行的字节码的行号指示器。

2.虚拟机栈：线程私有，存放基本数据类型、对象引用和 returnAddress 类型。

3.本地方法栈：为虚拟机使用到的 Native 方法服务。

4.Java 堆：线程共享，存放对象的实例，也是 GC 回收器管理的主要区域。

5.方法区：线程共享，存放已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。

6.运行时常量池：方法区的一部分，存放编译期生成的各种字面量和符号引用。

7.直接内存：不是虚拟机运行时数据区的一部分，也不是 Java 虚拟机规范中定义的内存区域，容易引起 OOM 异常，NIO 会调用，不受 Java 堆大小的限制。

16、Abstract 和 interface 区别？

回答：

1.abstract class 在 Java 语言中表示的是一种继承关系，一个类只能使用一次继承关系。但是，一个类却可以实现多个 interface。

2.在 abstract class 中可以有自己的数据成员，也可以有非 abstarct 的成员方法，而在 interface 中，只能够有静态的不能被

修改的数据成员（也就是必须是 static final 的，不过在 interface 中一般不定义数据成员），所有的成员方法都是 abstract 的。

3.abstract class 和 interface 所反映出的设计理念不同。其实 abstract class 表示的是"is-a"关系，interface 表示的是"like-a"关系。

4.实现抽象类和接口的类必须实现其中的所有方法。抽象类中可以有非抽象方法。接口中则不能有实现方法。

5.接口中定义的变量默认是 public static final 型，且必须给其初值，所以实现类中不能重新定义，也不能改变其值。

6.抽象类中的变量默认是 friendly 型，其值可以在子类中重新定义，也可以重新赋值。

7.接口中的方法默认都是 public,abstract 类型的。

17、有没有遇到过内存溢出，内存溢出怎么解决？

回答： 分不同情况解决，大多数情况下，此时如果代码没有问题的情况下，适当调整-Xmx 和-Xms 是可以避免的，不过一定是代码没有问题的前提，为什么会溢出呢，要么代码有问题，要么访问量太多并且每个访问的时间太长或者数据太多，导致数据释放不掉，因为垃圾回收器是要找到那些是垃圾才能回收，这里它不会认为这些东西是垃圾，自然不会去回收了。

18、jvm 调优有哪几种方式

回答：

1.观察内存释放情况、集合类检查、对象树

2.线程监控

3.内存泄漏检查

19、java 中 invokeAndWait 和 invokeLater 有什么区别？

回答：这两个方法是 Swing API 提供给 Java 开发者用来从当前线程而不是事件派发线程更新 GUI 组件用的。InvokeAndWait()同步更新 GUI 组件，比如一个进度条，一旦进度更新了，进度条也要做出相应改变。如果进度被多个线程跟踪，那么就调用 invokeAndWait()方法请求事件派发线程对组件进行相应更新。而 invokeLater()方法是异步调用更新组件的。

20、Swing API 中哪些方法是线程安全的？

回答：Swing 的规则是：一旦 Swing 组件被具现化（realized），所有可能影响或依赖于组件状态的代码都应该在事件派发线程中执行。所以有这 3 个线程安全的方法：repaint()，revalidate()，andinvalidate()。

21、如何在 Java 中创建 immutable 对象？

回答：

1.immutable 对象的状态在创建之后就不能发生改变，任何对它的改变都应该产生一个新的对象。

2.Immutable 类的所有的属性都应该是 final 的。

3.对象必须被正确的创建，比如：对象引用在对象创建过程中不能泄露(leak)。

4.对象应该是 final 的，以此来限制子类继承父类，以避免子类改变了父类的 immutable 特性。

5.如果类中包含 mutable 类对象，那么返回给客户端的时候，返回该对象的一个拷贝，而不是该对象本身（该条可以归为第一条中的一个特例）

22、Java 中的 readwritelock 是什么？

回答： 一般而言，读写锁是用来提升并发程序性能的锁分离技术的成果。Java 中的 ReadWriteLock 是 Java 5 中新增的一个接口，一个 ReadWriteLock 维护一对关联的锁，一个用于只读操作一个用于写。在没有写线程的情况下一个读锁可能会同时被多个读线程 持有。写锁是独占的，你可以使用 JDK 中的 ReentrantReadWriteLock 来实现这个规则，它最多支持 65535 个写锁和 65535 个读 锁。

23、多线程中的忙循环是什么？

回答： 忙循环就是程序员用循环让一个线程等待，不像传统方法 wait(), sleep() 或 yield() 它们都放弃了 CPU 控制，而忙循环不

会放弃 CPU，它就是在运行一个空循环。这么做的目的是为了保留 CPU 缓存，在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。

24、volatile 变量和 atomic 变量有什么不同？

回答：volatile 变量和 atomic 变量看起来很像，但功能却不一样。Volatile 变量可以确保先行关系，即写操作会发生在后续的读操作之前，但它并不能保证原子性。例如用 volatile 修饰 count 变量那么 count++ 操作就不是原子性的。而 AtomicInteger 类提供的 atomic 方法可以让这种操作具有原子性如 getAndIncrement()方法会原子性的进行增量操作把当前值加一，其它数据类型和引用变量也可以进行相似操作。

25、如果同步块内的线程抛出异常会发生什么？

回答： 这个问题坑了很多 Java 程序员，若你能想到锁是否释放这条线索来回答还有点希望答对。无论你的同步块是正常还是异常退出的，里面的线程都会释放锁，所以对比锁接口我更喜欢同步块，因为它不用我花费精力去释放锁，该功能可以在 finally block 里释放锁实现。

26、单例模式的双检锁式是什么？

回答： 双重检验锁模式（double checked locking pattern），是一

种使用同步块加锁的方法。程序员称其为双重检查锁，因为会有两次检查 instance == null，一次是在同步块外，一次是在同步块内。为什么在同步块内还要再检验一次？因为可能会有多个线程一起进入同步块外的 if，如果在同步块内不进行二次检验的话就会生成多个实例了。

27、如何在 Java 中创建线程安全的 singleton?

回答：

```
public class Singleton {
private static Singleton instance = null;
private Singleton() {}
public static synchronized Singleton getInstance(){
if (instance == null) instance = new Singleton();
return instance;
}
}
```

28、说说 jdk1.8 的新特性？

回答：

1.随着大数据的兴起，函数式编程在处理大数据上的优势开始体现，引入了 Lambada 函数式编程

2.使用 Stream 彻底改变了集合使用方式：只关注结果，不关心过程

3.新的客户端图形化工具界面库：JavaFX

4.良好设计的日期/时间 API

5.增强的并发/并行 API

6.Java 与 JS 交互引擎 -nashorn

7.其他特性

29、gc 回收机制原理？

回答：用户 Java 程序运行过程中，Java 虚拟机提供了另外一个系统级的线程，专门负责回收不再被使用的对象占用的内存，这一过程称为垃圾回收。垃圾回收需要对堆内存中的对象进行标记，并对堆内存进行整理。这一过程的某些阶段需要暂时终止用户 Java 线程，等回收工作完成后再恢复执行。因此，频繁地触发虚拟机垃圾回收操作的行为会影响程序的运行效率。那么什么情况下会频繁地出发垃圾回收操作呢？- 比如：堆内存设置过小- 再比如：程序频繁地分配大型局部对象数组。