1. 下面哪个不是死锁发生的必要条件：

A、互斥条件

B、阻塞并行

C、不可剥夺

D、循环等待

2．有关无等待说法错误的是：

A．无等待一定是无阻塞的

B．无等待是最高级别的无阻塞

C．无等待所有线程都必须在有限步内完成任务

D．无等待的性能是所有并行方法中最好的

3. 下面哪个是 Amdahl 定律的公式

A. 加速比=1/(F+1/n * (1-F))

B. 加速比=1/(F-1/n * (1-F))

C. 加速比=1/(F+1/n * (1+F))

D. 加速比=1/(F+1/n * (F-1))

4. 有关 Java 中线程的状态，说法错误的是：

A．线程启动后 start()，处于 Runnable 状态。

B. 线程执行结束后，不可以通过 start()方法，变成 Runnable 状态

C. 线程 synchronized 拿不到锁时，会变成 WAITING 状态

D．WAITING 状态的线程是可以被唤醒的

5. wait()和 notify()说法正确的是：

A．可以在任何场合随意调用 wait()方法

B．wait()方法会让线程 WAITING 等待，收到 notify 通知后，线程就可以立即继续执行

C．notfiy()从等待队列中随意选择一个线程进行通知，试图尝试让它继续执行。

D．notfiyAll()会通知所有等待在当前对象上的线程，因此所有线程都可以顺利继续往下执行。

6. Thread.join()有什么用

A．让两个线程合二为一

B． 让一个线程等待另外一个线程执行结束

C．让线程进行无条件休眠

D．让当前线程让出 CPU

7. 有关可见性问题的成因，说法不那么正确的是：

A. 编译器优化

B. 错误的应用层同步

C．多 CPU 缓存不一致

D．Java 虚拟机进行的编译优化

8. Happen-Before 规则中，说法错误的是：

A．程序的串行语义一致性

B．A 先于 B，B 先于 C，但 A 未必先于 C

C．线程的 start()方法先于它的每一个动作

D. 对象的构造函数执行结束先于 finalize()方法

9. CAS 说法不正确的是：

A．CAS 表示比较交换

B．CAS 是原子操作

C．CAS 是无障碍并行的核心方法

D．CAS 需要使用 CPU 汇编指令实现

10. AtomicInteger 说法正确的是

A．直接使用 AtomicInteger 不是线程安全的

B. 它的核心是锁

C. 它不会产生饥饿

D. 至少有一个线程可以在有限步内完成操作

11. AtomicInteger 的 incrementAndGet 方法正确的一种实现应该是：

A.

```
public final int incrementAndGet() {
    while(true) {
        int current = get();
        int next = current + 1;
        if (compareAndSet(current, next))
            return next;
    }
}
```

B.

```
public final int incrementAndGet() {
    int current = get();
    int next = current + 1;
    if (compareAndSet(current, next))
        return next;
}
```

C.

```
public final int incrementAndGet() {
    for(;;) {
        if (compareAndSet(get(), get()+1))
            return get();
    }
}
```

D.

```
public final int incrementAndGet() {
    for(;;) {
        int current = get();
        int next = get() + 1;
        if (compareAndSet(current, next))
            return next;
    }
}
```

12. AtomicStampedReference 说法错误的是

A．通过引入一个不重复的 long 型标记，解决 ABA 问题
B．它是一个模板类，可以保存各种类型的数据
C．内部实现通过一对 Pair，来封住一个引用和 Stamped 数据
D． 如果忽略 Stamped，它就退化为 AtomicReference 的功能

13. AtomicIntegerArray 说法不正确的是：
A. 它封装了对 int 数组的 CAS 操作
B. 内部存放数据的场所就是 int[]
C. 可以线程安全地修改数组引用本身
D．内部使用偏移量进行数组内元素的定位

14. AtomicIntegerFieldUpdater 说法正确的是：
A．可以加速对 AtomicInteger 的修改，提高性能
B．可以对任何字段进行 CAS 操作
C．操作的数据类型必须是 volatile
D．支持 static 变量

15. 有关 ReentrantLock 说法错误的是：
A．可以通过 ReentrantLock，实现公平锁
B．它可以支持中断和超时
C．它内部实现通过 CAS 来进行
D．它的性能要比内部锁好很多

16. 有关读写锁，说法错误的是：
A．读锁不会阻塞写锁，因此读写锁性能一般要比重入锁好
B．写锁之间会阻塞
C．区分读锁和写锁的主要目的就是为了进行锁分离
D．在 JDK5 之后，就可以使用读写锁了

17.有关 LockSupport.park()错误的是：
A．让线程继续停止执行
B． 和 suspend 一样，是不推荐使用的，因为容易引起线程冻结
C． 对应的操作是 unpark()让线程继续执行，类似于 resume()
D．被 park()的线程如果遇到中断，不会抛出异常。

18. 有关 BlockingQueue 说法错误的是：
A．这是一个接口，JDK 内部有多种不同的实现
B．插入数据时，如果队列满了，线程就会阻塞
C． 这是一个高并发的实现，性能不错
D． ArrayBlockingQueue 是 BlockingQueue 的一种实现，内部使用数组

19. 有关 JDK 的内置线程池，说法错误的是：
A．固定大小的线程池总是拥有给定的线程数量
B．CachedThreadPool 线程数量可以自由伸缩

C．如果需要的线程数量超过 coreSize，线程池就会将线程数量扩张，但不超过 MaxSize

D．JDK 中线程池所属的最重要的接口是 Executor，它拥有一个 execute()方法。

20. 有关 ForkJoinPool 说法错误的是：

A．fork()会创建一个新的任务 推入线程池

B．ForkJoinPool 中重要的 2 个接口是 RecursiveAction 和 RecursiveTask，分别用于不需要返回值，和需要返回值的场景

C．ForkJoinPool 内部会维护一个无锁的 Queue，保存休眠的线程

D．为了提高性能，ForkJoinPool 采用工作窃取的方式，允许线程之间相互帮助

21. 下面有 4 个单例的实现，哪个你认为是最不靠谱的：

A.

```
public class Singleton {
    private Singleton(){
        System.out.println("Singleton is create");
    }
    private static Singleton instance = new Singleton();
    public static Singleton getInstance() {
        return instance;
    }
}
```

B.

```
public class LazySingleton {
    private LazySingleton() {
        System.out.println("LazySingleton is create");
    }
    private static LazySingleton instance = null;
    public static synchronized LazySingleton getInstance() {
        if (instance == null)
            instance = new LazySingleton();
        return instance;
    }
}
```

C.

```
public class StaticSingleton {
    private StaticSingleton(){
        System.out.println("StaticSingleton is create");
    }
    private static class SingletonHolder {
        private static StaticSingleton instance = new StaticSingleton();
    }
    public static StaticSingleton getInstance() {
```

```
            return SingletonHolder.instance;
        }
}
```

D.

```
public class Singleton {
    private static volatile Singleton instance;

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

    private Singleton() throws IOException {
    }
}
```

22. 上述 21 题里面 4 种单例的实现，哪个你认为是最好的？

23. 不变模式说法：

1）. 不变模式的对象实例创建之后，就一定不能再改变了。

2）. 不变模式内部，可以使用 final 来保证字段的不变性。

3）. java.lang.String 是不变的，Integer 也是不变的。

4）. 不变模式最大的好处是不用担心多线程访问的不一致，因此，不需要进行同步。

上述说法正确的个数是：

A． 1 个 B. 2 个 C. 3 个 D.4 个

24. 有关 Buffer 说法错误的是：

A． Buffer 中的 position 表示当前位置

B． Buffer 中的 capacity 表示缓冲区实际上限

C． Buffer 中的 flip()操作通常用在读写转换上

D． Buffer 通常和通道 Channel 相连接

25. 对于 NIO 的 selector 及其相关操作，说法错误的是：

A. selector 使用线程复用，一个线程可以管理多个 Channel

B．当数据读取完毕后，selector 的 select()方法就能收到通知

C．即使是所谓的非阻塞网络编程，实际使用中，还是会有阻塞发生的

D．Channel 也可以工作在阻塞模式

26. 有关锁优化，说法错误的是：
A．减少锁的持有时间，在大部分情况下，可以提供并行程序性能
B．减小锁粒度的一个典型应用场景是 Collections.SynchronizedMap
C. 锁粗化的思想和减少锁持有时间相反
D．LinkedBlockingQueue 使用了锁分离的思想，take()和 put()分别在队列两端进行，没有冲突

27. 进行锁消除的必要条件不包括：
A．开启了逃逸分析-XX:+DoEscapeAnalysis
B．开启了锁消除-XX:+EliminateLocks
C．关闭偏向锁
D．变量实际上没有发生逃逸

28. 有关偏向锁说法错误的是：
A．偏向锁总是可以提升性能，因此 JVM 默认会启动偏向锁
B．打开偏向锁使用 -XX:+UseBiasedLocking
C．只要没有竞争，获得偏向锁的线程，在将来进入同步块，不需要做同步
D．当有其它线程请求相同的锁时，偏向模式宣告结束

29.有关 JVM 自旋锁说法错误的是：
A．自旋锁会做一些空循环，等待一段时间后再尝试拿锁
B．如果自旋锁成功拿到锁，就可以避免线程挂起，从而提高性能
C．在 JDK 1.7 中，你可以关闭自旋锁
D．快进快出同步块，可以提供自旋成功率

30. LongAdder 说法错误的是：
A．内部使用的 CAS 操作
B．LongAdder 会进行冲突的自动调整，如果发现冲突，会扩展数据槽位
C．LongAdder 的问题在于即使没有冲突，还是会进行热点分离，导致性能下降
D．LongAdder 在 JDK 8 中才可以使用

31. 有关 StampedLockd 的说法错误的是：
A．可以理解是读写锁的扩展
B．一个重要的核心是支持乐观读
C．乐观读是无障碍的
D. 乐观读的性能总是比悲观读好