**中软国际考题-笔试**

1. 单选题

1、启动多线程的方式是调用如下哪一个方法（C）

A.start B.do C.run D.sleep

2、Java提供了一个\_\_\_\_\_\_\_\_\_\_线程，自动回收动态分配的内存。（D）

A．异步 B．消费者 C．守护 D．垃圾收集

3、当▁方法终止时，能使线程进入死亡状态。（A）

A．run B．setPrority//更改线程优先级

C．yield//暂停当前线程的执行 执行其他线程 D．sleep//线程休眠

4、线程通过▁▁方法可以使具有相同优先级线程获得处理器。（C）

A．run B．setPrority C．yield D．sleep

5、哪个关键字可以对对象加互斥锁？  ( B )

   A．transient  B．synchronized  C．serialize  D．static

6. 以下哪个方法用于定义线程的执行体？( C  )

A、 start() B、init() C、run() D、synchronized()

7、线程通过（ D ）方法可以休眠一段时间，然后恢复运行。

A. run B. setPrority C. yield D. sleep

8、resume() 方法负责恢复哪些线程的执行？（B）

1. 通过调用 wait() 方法而停止运行的线程
2. 通过调用 suspend() 方法而停止运行的线程
3. 通过调用 sleep() 方法而停止运行的线程
4. 通过调用 stop() 方法而停止运行的线程

9、运行在操作系统上的每个应用都会独占一个独立的？（B）

A.线程 B.process C.thread D.Runnable

10、下面关于线程的叙述中，正确的是（C）

A  不论是系统支持线程还是用户级线程，其切换都需要内核的支持

B  线程是资源的分配单位，进程是调度和分配的单位

C  不管系统中是否有线程，进程都是拥有资源的独立单位

D  在引入线程的系统中，进程仍是资源分配和调度分派的基本单位

11、下列哪个关键字可以用于实现同步?(C )

(A) Native (B) static (C) Synchronized (C) abstract

12、请问wait()方法在以下哪个代码中被调用？（C）

A.一个while()循环体中 B.run方法中 C.同步化代码块中 D. 代码的任何地方

1. 多选题
2. 以下关于ThreadLocal的作用描述正确的是？（A、D、E）
3. ThreadLocal用来解决多线程程序的并发

B. 多线程类需要继承ThreadLocal类;

C. 多线程类需要实现ThreadLocal接口

1. ThreadLocal并不是一个Thread,而是Thread的局部变量
2. ThreadLocal为每个使用该变量的线程提供独立的变量副本
3. 在java中实现多线程的方式有哪几种？（A、D）
4. 继承Thread类 B. 调用sleep方法等待; C. 继承Runnable类
5. 实现Runnable接口 E. 实现Thread接口

3、有三种原因可以导致线程不能运行，它们是▁。（A、B、C）

A．等待 B．阻塞 C．休眠 D．挂起及由于I/O操作而阻塞

4、下列关于Java多线程并发控制机制的叙述中，错误的是：(B C )

A、Java中对共享数据操作的并发控制是采用加锁技术

B、线程之间的交互，提倡采用suspend()/resume()方法

C、共享数据的访问权限都必须定义为private

D、Java中没有提供检测与避免死锁的专门机制，但应用程序员可以采用某些策略防止死锁的发生

5、关于sleep()和wait()，以下描述正确的是（ A、B、C ）

A. sleep是线程类（Thread）的方法，wait是Object类的方法；

B. sleep不释放对象锁，wait放弃对象锁；

C. sleep暂停线程、但监控状态仍然保持，结束后会自动恢复；

D. wait后进入等待锁定池，只有针对此对象发出notify方法后获得对象锁进入运行状态。

6、有关线程的哪些描述是正确的？（B、C）

A. 一旦一个线程被创建，它就立即开始运行

B. 使用 start() 方法可以使一个线程成为可运行的，但是它不一定立即开始运行

C. 一个线程可能因为不同的原因停止 (cease) 并进入就绪状态

D. 当一个线程因为抢先机制而停止运行时，它被放在可运行队列的前面

7、 以下哪几种是Java实现多线程的方式？（A、C）

A. Thread类 B. Runnable类 C. Runnable接口

D. Thread接口

8、有三种原因可以导致线程不能运行，它们是 （ A B C ）

A．等待 B．阻塞 C．休眠 D．挂起及由于I/O操作而阻塞

9、 下列哪些方法可以启动线程？（A、c）

A. start()方法 B.go() C. run()方法

D. First（）方法

10、final、finally和finalize的区别（A、B、C）

A. final是修饰符，可以修饰类、方法、常量

B. finally在异常处理时的操作，可以提供finally来执行清除操作

C. finalize是方法名，在垃圾收集器在对象从内存中清除之前做必要的清理工作

D. finally和finalize都是异常处理的方法

1. 填空题
2. 如果线程类实现了Runnable接口，则需要实现该接口的\_\_\_\_\_\_\_\_\_\_ 方法，在启动线程时需要调用\_\_\_\_\_\_\_\_\_方法。(run、start)
3. Java中的线程有四种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、结束。(运行、阻塞)
4. 对处于可运行状态的多个线程对象进行系统级的协调，防止多个线程争用有限资源而导致系统\_\_\_\_\_\_\_\_\_\_。(死机或者崩溃)
5. java 的调度策略是基于线程的\_\_\_\_\_\_\_\_\_\_抢先式调度。意思就是，谁的优先级高那我就先给谁使用系统资源。（优先级）
6. 创建线程时需要实现Runnable接口
7. Java中的线程有四种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、结束。(运行、阻塞)
8. 线程同步的关键字是：\_\_\_\_\_\_\_\_\_\_ 。 (synchronized )
9. Java中的线程有五种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、终止。(就绪、运行、阻塞)
10. 线程同步与通讯有哪几种方式：信号量、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_。(Lock接口，同步关键字)
11. Java中的线程有四种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、结束。(运行、阻塞)
12. Java中的线程有四种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、结束。(运行、阻塞)
13. Java中的线程有四种状态分别是：新建、\_\_\_\_\_\_\_\_\_\_ 、\_\_\_\_\_\_\_\_\_、结束。(运行、阻塞)
14. 判断题
15. ArrayList、Vector、Map等对象都是线程安全的。（ F ）
16. 在Java中，高优先级的可运行线程会抢占低优先级线程。（T ）
17. 线程使用sleep方法去休眠后可以使用notify方法唤醒（ F）
18. 线程使用sleep方法休眠是会释放该线程同步锁定的对象。( F )
19. 在Java中，高优先级的可运行线程会抢占低优先级线程。（T ）
20. 线程可以用yield方法使低优先级的线程运行。（F）
21. 线程可以用yield方法使低优先级的线程运行。（ F ）
22. 在Java中，高优先级的可运行线程会抢占低优先级线程。（ T ）
23. 一个线程在调用它的 start() 方法之前，该线程将一直处于新建状态。（ T ）
24. 一个线程可以调用 yield() 方法使其他线程有机会运行。（ T ）
25. 如果线程终止，它便不能运行。（ T ）
26. 多个程序同时运行使用的是时间片轮转算法。（ T ）
27. 当今的操作系统绝大部分是基于单任务的操作系统（F）
28. 线程池中的所有线程都是处于时时刻刻唤醒状态（F）
29. 如果线程死亡，它便不能运行。（T）
30. Stop()方法是终止当前线程的一种状态( T )
31. 互斥条件：一个资源每次只能被一个进程使用。（ T ）
32. 多线程是Java语言独有的。（ F ）
33. 多线程需要多CPU。（ F ）

五、简答题

1、什么是线程池？ 为什么要使用它？

答案：

1. 创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。
2. 为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，
3. 里面的线程叫工作线程。
4. 从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

2、什么是线程安全？

答案：

1. 就是线程同步的意思，就是当一个程序对一个线程安全的方法或者语句进行访问的时候，其他的不能再对他进行操作了，必须等到这次访问结束以后才能对这个线程安全的方法进行访问
2. 如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。
3. 若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则就可能影响线程安全。

3、什么是线程？

答案：线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对 运算密集型任务提速。比如，如果一个线程完成一个任务要100毫秒，那么用十个线程完成改任务只需10毫秒。Java在语言层面对多线程提供了卓越的支持，它也是一个很好的卖点。

4、Runnable接口和Callable接口的区别？

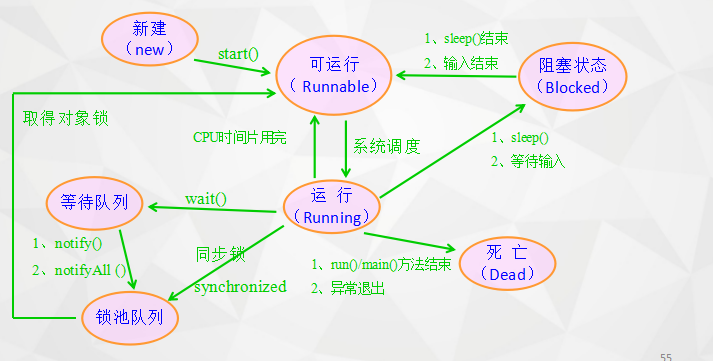
答案：Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行run()方法中的代码而已；Callable接口中的call()方法是有返回值的，是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。

5. start()方法和run()方法的区别

答案：只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

6、画图描述线程状态转换（5分）

答案：



7、试简述Thread类的子类或实现Runnable接口两种方法的异同？

答案：

（1）继承 Thread 类，实现步骤

1）自定义一个类，继承自 Thread 类，然后重写里面的run方法

2）在主线程中创建自定义类的实例对象，通过调用 start() 方法启动线程

（2）实现 Runnable 接口，步骤

1）自定义一个类实现 Runnable 接口并重写里面的 run() 方法

2）在主线程中创建此类的实例对象

3）创建 Thread 类的实例对象，将自定义类的实例对象作为参数进行传递

4）通过 start() 方法启动线程

共同点：子类重写 run() 方法，但是继承是重写 Thread 中的 run() 方法，实现接口是 Runnable 中的 run() 方法，而且都需要 start() 方法来启动线程。

不同点：继承是通过其创建的子类对象来调用 start() 方法的，接口是将子类对象作为参数来传递给 Thread ，然后通过 Thread 的对象来调用 start() 启动的。

另外，实现 Runnable 接口还具有以下功能：

1) 适合多个相同程序代码的线程处理同一资源的情况，可以把线程同程序中的数据有效分离，较好地体现了面向对象的设计思想。

2) 可以避免由于Java的单继承特性带来的局限。

3) 增强了代码的健壮性，代码能够被多个线程共同访问，代码与数据是独立的。多个线程可以操作相同的数据，与它们的代码无关。当线程被构造时，需要的代码和数据通过一个对象作为构造函数实参传递进去，这个对象就是一个实现了 Runnable 接口的类的实例。

8、 Java中 Runnable 和 Callable 有什么不同?

答案：

在实现多线程时，两种不同的方法分别用到了这两种接口。当实现 Runnable 接口时，重写里面的 run 方法拥有给虚拟机调用，在实现 Callable 时，重写里面的 call() 方法用于虚拟机调用。

在主线程中启动时，Runable 接口是通过将其子类对象传入 Thread 类，然后通过 Thread 类的对象调用 start() 方法实现的。而 Callable 接口是通过线程池对象调用 submit() 方法启动的，然后虚拟机调用 call() 方法实现的。其中 call() 方法可以有返回值，run() 方法没有。call() 可以抛出受检查的异常，比如 ClassNotFoundException ， 而 run() 不能抛出受检查的异常。

9、 请简述ThreadLocal 的作用是什么？

答案：

1）ThreadLocal 用来解决多线程程序的并发问题。

2）ThreadLocal 并不是一个 Thread ，而是 Thread 的局部变量，当使用 ThreadLocal 维护变量时，ThreadLocal 为每个使用该变量的线程提供独立的变量副本，所以每个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。

3）从线程的角度看，目标变量就象是线程的本地变量，这也是类名中 Local 所要表达的意思。

4）线程局部变量并不是 Java 的新发明，Java 没有提供在语言级支持(语法上)，而是变相地通过 ThreadLocal 的类提供支持。

10、 请简述 sleep() 方法和 yield() 方法的区别。

答案：

1) sleep() 方法暂停当前线程后，会给其他线程执行机会，线程优先级对此没有影响。

yield()方法会给优先级相同或更高的线程更高的执行机会。

2) sleep() 方法会将线程转入阻塞状态，直到阻塞时间结束，才会转入就绪状态。

yield() 方法会将当前线程直接转入就绪状态。

3) sleep() 方法声明抛出了 InterruptedException 异常，所以调用 sleep() 方法时要么捕捉该异常，要么显示声明抛出该异常。

yield() 方法则没有声明抛出任何异常。

4) sleep() 方法比 yield() 方法有更好的移植性，通常不建议使用 yield() 方法来控制并发线程的执行。

11、 请说明什么是乐观锁和悲观锁？

答案：

1）乐观锁：对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-设置这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

2）悲观锁：对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，直接对操作资源上了锁。

12、 试说明在Java中 wait() 和 sleep() 方法的不同。

答案：

区别1：wait() 方法是在 Object 类中，而 sleep() 方法是 Thread 类中。

区别2：sleep() 方法导致了程序暂停执行指定的时间，让出 cpu 给其他线程，但是他的监控状态依然保持着，当指定的时间到了又会自动恢复运行状态。在调用 sleep() 方法的过程中，线程不会释放对象锁。

而当调用 wait() 方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用 notify() 方法后本线程才进入对象锁定池准备，获取对象锁进入运行状态。

13、1、 什么是线程？

答案：线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要100毫秒，那么用十个线程完成改任务只需10毫秒。Java在语言层面对多线程提供了卓越的支持，它也是一个很好的卖点

14、 线程和进程有什么区别？

答案：线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

15、 如何在Java中实现线程？

答案：1.java.lang.Thread 类的实例就是一个线程但是它需要调用java.lang.Runnable接口来执行，

2.由于线程类本身就是调用的Runnable接口所以你可以继承java.lang.Thread 类或者直接调用Runnable接口来重写run()方法实现线程。

3.实现Callable接口通过FutureTask包装器来创建Thread线程

16、 Thread 类中的start() 和 run() 方法有什么区别

1.start（）方法来启动线程，真正实现了多线程运行。这时无需等待run方法体代码执行完毕，可以直接继续执行下面的代码；通过调用Thread类的start()方法来启动一个线程， 这时此线程是处于就绪状态， 并没有运行。 然后通过此Thread类调用方法run()来完成其运行操作的， 这里方法run()称为线程体，它包含了要执行的这个线程的内容， Run方法运行结束， 此线程终止。然后CPU再调度其它线程。

2.run（）方法当作普通方法的方式调用。程序还是要顺序执行，要等待run方法体执行完毕后，才可继续执行下面的代码； 程序中只有主线程——这一个线程， 其程序执行路径还是只有一条， 这样就没有达到写线程的目的。

记住：多线程就是分时利用CPU，宏观上让所有线程一起执行 ，也叫并发

17、 什么是线程安全？Vector是一个线程安全类吗？

答案：如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector 是用同步方法来实现线程安全的, 而和它相似的ArrayList不是线程安全的。

18、 Java中notify 和 notifyAll有什么区别？

答案：notify()&notifyall()的共同点：均能唤醒正在等待的线程，并且均是最后只有一个线程获取资源对象的锁。

不同点：notify() 只能唤醒一个线程，而notifyall()能够唤醒所有的线程，当线程被唤醒以后所有被唤醒的线程竞争获取资源对象的锁，其中只有一个能够得到对象锁，执行代码。

注意：wait()方法并不是在等待资源的锁，而是在等待被唤醒（notify()），一旦被唤醒后，被唤醒的线程就具备了资源锁（因为无需竞争），直至再次执行wait()方法或者synchronized代码块执行完毕。

19、 什么是线程池？ 为什么要使用它？

答案：创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

20、 有三个线程T1，T2，T3，怎么确保它们按顺序执行

答案：在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3调用T2，T2调用T1)，这样T1就会先完成而T3最后完成。

#### 21、如何在Java中实现线程？

extends java.lang.Thread 重写run()

implements java.lang.Runnable接口 重写run()

22、在Java中wait()和sleep()方法的不同?

区别1：wait方法是在Object类中，而sleep方法是Thread类中

区别2：sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用sleep()方法的过程中，线程不会释放对象锁。

而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备，获取对象锁进入运行状态。

23、线程和进程有什么区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

24、Thread 类中的start() 和 run() 方法有什么区别？

start()方法被用来启动新创建的线程，而且start()内部 调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启 动，start()方法才会启动新线程。

25、wait()方法和sleep()方法有什么区别？

答案：对于sleep()方法，我们首先要知道该方法是属于Thread类中的。而wait()方法，则是属于Object类中的。

sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。

在调用sleep()方法的过程中，线程不会释放对象锁。

而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备获取对象锁进入运行状态。

26. 在Java中Lock接口比synchronized块的优势是什么？

答案：lock接口在多线程和并发编程中最大的优势是它们为读和写分别提供了锁，它能满足你写像ConcurrentHashMap这样的高性能数据结构和有条件的阻塞。

27. ThreadLocal的用途?

答案：1)ThreadLocal用来解决多线程程序的并发问题

2)ThreadLocal并不是一个Thread,而是Thread的局部变量,当使用ThreadLocal维护变量时,ThreadLocal为每个使用该变量的线程提供独立的变量副本,所以每个线程都可以独立地改变自己的副本,而不会影响其它线程所对应的副本.

3)从线程的角度看，目标变量就象是线程的本地变量，这也是类名中“Local”所要表达的意思。

4)线程局部变量并不是Java的新发明,Java没有提供在语言级支持(语法上),而是变相地通过ThreadLocal的类提供支持

28.volatile关键字是否能保证线程安全？

答案：不能。volatile关键字用在多线程同步中，可保证读取的可见性，JVM只是保证从主内存加载到线程工作内存的值是最新的读取值，而非cache中。但多个线程对volatile的写操作，无法保证线程安全。例如假如线程1，线程2 在进行read,load 操作中，发现主内存中count的值都是5，那么都会加载这个最新的值，在线程1堆count进行修改之后，会write到主内存中，主内存中的count变量就会变为6；线程2由于已经进行read,load操作，在进行运算之后，也会更新主内存count的变量值为6；导致两个线程及时用volatile关键字修改之后，还是会存在并发的情况。

六、程序实现题

1、信号量问题：有时被称为信号灯，是在多线程环境下使用的一种设施, 它负责协调各个线程, 以保证它们能够正确、合理的使用公共资源，假设初始化公共资源的数量为n，当需要获取公共资源时需要申请，当剩余可以使用的公共资源时，则分发，如当前无可使用的公共资源则等待。若某一个线程使用公共资源后可以释放，并交由其他线程使用。

（10分）

|  |
| --- |
| package test;  import java.util.Random;  import java.util.Vector;  public class Semaphore {  private Vector<Object> locks = new Vector<Object>();  private int permitNum = 1;  private int nowPermitNum = 1;  private boolean permitNumGrow = false;  private boolean fair = false;  Random random = new Random();  public Semaphore(int permitNum, boolean permitNumGrow, boolean fair) {  this.permitNum = permitNum;  this.nowPermitNum = permitNum;  this.permitNumGrow = permitNumGrow;  this.fair = fair;  }  public Semaphore(int permitNum) {  this(permitNum, true, false);  }  public Semaphore() {  this(1);  }  public void acquire() {  Object lock = new Object();  synchronized (lock) {  if (nowPermitNum > 0) {  nowPermitNum--;  } else {  locks.add(lock);  try {  lock.wait();  } catch (Exception ex) {  ex.printStackTrace();  }  }  }  }  public void release() {  if (locks.size() > 0) {  int index = 0;  if (!fair) {  index = Math.abs(random.nextInt()) % locks.size();  }  Object lock = locks.get(index);  locks.remove(lock);  synchronized (lock) {  lock.notify();  }  } else if (nowPermitNum < permitNum || permitNumGrow) {  nowPermitNum++;  }  }  public int getAvailablePermitNum() {  return nowPermitNum;  }  } |

2、有一个抽奖池,该抽奖池中存放了奖励的金额,该抽奖池用一个数组int[] arr = {10,5,20,50,100,200,500,800,2,80,300};

创建两个抽奖箱(线程)设置线程名称分别为“抽奖箱1”，“抽奖箱2”，随机从arr数组中获取奖项元素并打印在控制台上,格式如下:

抽奖箱1 又产生了一个 10 元大奖

抽奖箱2 又产生了一个 100 元大奖

（10分）

|  |
| --- |
| public class ChoujiangDemo {  public static void main(String[] args) {  Chou c = new Chou();  Thread t1 = new Thread(c,"抽奖箱1");  Thread t2 = new Thread(c,"抽奖箱2");  t1.start();  t2.start();  }  }  public class Chou implements Runnable {  int[] arr = { 10, 5, 20, 50, 100, 200, 500, 800, 2, 80, 300 };  int num = arr.length;  boolean[] flag = new boolean[arr.length];  public void run() {  while (true) {  synchronized (this) {  if (num > 0) {  int index = (int) (Math.random() \* arr.length);  int get = arr[index];  // 代表这张抽奖券抽过了  if (flag[index] != true) {  flag[index] = true;  System.out.println(Thread.currentThread().getName()  + " 又产生了一个" + get + "元大奖");  num--;  }  }  }  }  }  } |

3、多线程实现多个窗口同时卖票

public class SellTicktDemo implements  Runnable {

    //定义票的总数

    private int total = 100;

    //定义票的编号

    private int no = total+1;

    //定义一个线程同步对象

    private Object obj = new Object();

    @Override

    public void run() {

        while(true){

            //同步锁

            synchronized(this.obj){

                if(this.total > 0){

                    try {

                        Thread.sleep(100);

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                    String msg = Thread.currentThread().getName()+" 售出第   "+(this.no -this.total) +"  张票";

                    System.out.println(msg);

                    this.total--;

                }else{

                    System.out.println("票已售完，请下次再来！");

                    System.exit(0);

                }

            }

        }

    }

}

测试类：

public class SellTicktDemoTest {

    public static void main(String[] args) {

        //得到对象

        SellTicktDemo std = new SellTicktDemo();

        //把对象放入线程中

        Thread t1 = new Thread(std,"售票窗口1");

        Thread t2 = new Thread(std,"售票窗口2");

        Thread t3 = new Thread(std,"售票窗口3");

        Thread t4 = new Thread(std,"售票窗口4");

        t1.start();

        t2.start();

        t3.start();

        t4.start();

    }

}

4、[编写多线程应用程序，模拟多个人通过一个山洞的模拟。这个山洞每次只能通过一个人，每个人通过山洞的时间为5秒，随机生成10个人，同时准备过此山洞，显示一下每次通过山洞人的姓名。](http://www.cnblogs.com/wenwen123/p/5740579.html)

（10分）

|  |
| --- |
| public class ShanDong extends Thread{  @Override  public void run() {  synchronized (this) {  System.out.println(Thread.currentThread().getName()+"正在过山洞,请等待5秒");  try {  Thread.sleep(5000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }    }    public static void main(String[] args) {  String[] peoples = {"赵","钱","孙","李","周","吴","郑","王","冯","陈"};  ShanDong sd = new ShanDong();  for(String people:peoples){  Thread t = new Thread(sd,people);  t.start();  }    }  } |

5、请编写多线程的程序模拟三个售票窗口同时出售20张票。

**Station.java**

|  |
| --- |
| **public** **class** Station **extends** Thread {  // 通过构造方法给线程名字赋值  **public** Station(String name) {  **super**(name);// 给线程名字赋值  }  // 为了保持票数的一致，票数要静态  **static** **int** *ticketCount* = 20;  // 创建一个静态钥匙  **static** Object *objLock* = "ticketLock";// 值是任意的  // 重写run方法，实现买票操作  @Override  **public** **void** run() {  **while** (*ticketCount* > 0) {  **synchronized** (*objLock*) {// 这个很重要，必须使用一个锁，  // 进去的人会把钥匙拿在手上，出来后才把钥匙拿让出来  **if** (*ticketCount* > 0) {  System.***out***.println(getName() + "卖出了第" + *ticketCount* + "张票");  *ticketCount*--;  } **else** {  System.***out***.println("票卖完了");  }  }  **try** {  *sleep*(1000);// 休息一秒  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  } |

**MainApp.java**

|  |
| --- |
| **public** **class** MainApp {  /\*\*  \* java多线程同步锁的使用 示例：三个售票窗口同时出售20张票  \*/  **public** **static** **void** main(String[] args) {  // 实例化站台对象，并为每一个站台取名字  Station station1 = **new** Station("窗口1");  Station station2 = **new** Station("窗口2");  Station station3 = **new** Station("窗口3");  // 让每一个站台对象各自开始工作  station1.start();  station2.start();  station3.start();  }  } |

**参考运行结果：**

**6、编写线程打印5次，每次睡眠1秒**

（10分）

|  |
| --- |
| public class NameRunnable implements Runnable {  public void run() {  for(int i=0;i<5;i++){  System.out.println(Thread.currentThread().getName());  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  } |

7、写两个线程，一个线程打印1~52，另一个线程打印A~Z，打印顺序是12A34B...5152Z；

（10分）

|  |
| --- |
| public class TwoThread {    public static void main(String args[]){  MyObject1 my = new MyObject1();  new Thread(new Runnable(){  @Override  public void run() {  // TODO Auto-generated method stub  for(int i = 0; i < 26; i++){  my.printNum();  }  }    }).start();  new Thread(new Runnable(){  @Override  public void run() {  // TODO Auto-generated method stub  for(int i = 0; i < 26; i++){  my.printA();  }  }    }).start();  }  }class MyObject1{  private static boolean flag = true ;  public int count = 1;    public synchronized void printNum(){  while(flag == false){  try {  this.wait();  } catch (InterruptedException e) {  // TODO Auto-generated catch block e.printStackTrace();  }  }  System.out.print((2\*count-1));  System.out.print(2\*count);    flag = false;  this.notify();  }  public synchronized void printA(){  while(flag == true){  try {  this.wait();  } catch (InterruptedException e) {  // TODO Auto-generated catch block e.printStackTrace();  }  }  System.out.print((char)(count+'A'-1));  count++;  flag = true;  this.notify();  }  }  [IMG_256](https://www.cnblogs.com/zhihuayun/p/javascript:void(0);) |

8、子线程循环10次，主线程循环100次，然后子线程循环10次，主线程循环100次，这样循环50次；

|  |
| --- |
| public class SubMain {      public static void main(String args[]){  MyObject2 m = new MyObject2();  new Thread(new Runnable(){  @Override  public void run() {  // TODO Auto-generated method stub  for(int i = 0; i<5; i++)  m.sub();  }    }).start();    for(int i = 0; i<5; i++)  m.main1();  }  }  class MyObject2{  private boolean flag = true;  //flag是true时执行sub方法，flag是false时执行main1方法  public synchronized void sub(){  while(flag == false){ //如果flag==false，说明另一个线程拥有该对象的锁，调用sub的方法被阻塞，直到另一个线程释放锁，唤醒该线程。  try {  this.wait();  } catch (InterruptedException e) {  // TODO Auto-generated catch block e.printStackTrace();  }  }    for(int i = 0 ; i<10; i++){  System.out.print("s");  }  System.out.println();  flag = false;  this.notify();  }    public synchronized void main1(){  while(flag == true){  try {  this.wait();  } catch (InterruptedException e) {  // TODO Auto-generated catch block e.printStackTrace();  }  }    for(int i = 0 ; i<100; i++){  System.out.print("m");  }  System.out.println();  flag = true;  this.notify();  }    } |

9、需求说明：定义一个线程，输出1 ～ 200之间的奇数，定义另一个线程，逆序输出1 ～ 200之间的奇数。要求：采用Thread类和Runnable接口两种方式实现，要求线程A和线程B交替，使用sleep()方法阻塞当前线程，每打印一次后休息1秒。

|  |
| --- |
| public class AThread extends Thread{  @Override  public void run() {  // TODO Auto-generated method stub  //super.run();  for(int i=1;i<=200;i++){  If(i%2!=0){  System.out.println(getName()+":i="+i);  }  Sleep(1000);  }  }  }  public class BThread extends Thread{  @Override  public void run() {  // TODO Auto-generated method stub  //super.run();  for(int i=200;i>=1;i--){  If(i%2!=0){  System.out.println(getName()+":i="+i);  }  Sleep(1000);  }  }  }  public class Test {  public static void main(String[] args) {  // TODO Auto-generated method stub  AThread th1 = new AThread();  BThread th2 = new BThread();    th1.start();  th2.start();  }  } |