# 同步练习题

1. 指出与线程相关的三个问题

（1）.竞态条件，读脏数据即不是最新的数据（2）.数据竞争，多个线程读取到了同一数据，但处理后却有多种结果（3）.缓存变量，读取到还未存入内存的缓存数据之前的数据。

1. 判断对错：当程序计算的正确性取决于相对时间或者调度器所控制的多线程交叉时，你会遇到数据竞争问题。

错误，这是竞态竞争，数据竞争要多个线程写入数据的竞争导致结果不同

1. 给出同步的定义

多个线程同时读取一个数据，当某线程中对其进行修改并写后其它线程中该数据需要更新至最新数据。

//指的是一种JVM特性，它用于保证两条及两条以上的线程不会同时在一个临界区中执行。

1. 指出同步的两种属性

//互斥性和可见性

1. 同步是如何实现出来的

//是基于监听器实现出来的，它是控制对临界区进行访问的并发构造，必须不可分割地执行。每个Java对象都关联着一个监听器，线程可以通过上锁或解锁的方式获取或者释放监听器上的锁。

1. 判断对错：当一个已经获取锁的线程调用任意Thread的任意sleep（）方法时不会释放锁。

对。

1. 如何指定一个同步方法

方法定义中加入关键字synchronized关键字

1. 如何指定一个同步块

synchronized（data）{} 对data读写挂锁的同步块

1. 给出活跃性的定义

正确的事情最终会发生

1. 指出三种活跃性挑战

（1）.死锁（2）.活锁（3）.饿死

1. 如何区分volatile和synchronized关键字？

Volatile只能保证数据读写直接进入内存，解决可见性问题；synchronized则是保证在其作用域中只能有一个线程运行，解决互斥性

1. 判断对错：java让你能够安全的访问一个final的属性而无需进行同步。对，final定义的数据在初始化后不允许进行修改。
2. 指出下面CheckingAccount类存在的线程问题：

**public class** CheckingAccount {  
 **private int balance**;  
 **public** CheckingAccount(**int** initialBalance){  
 **balance** = initialBalance;  
 }  
 **public boolean** withdraw(**int** amount){  
 **if**(amount<=**balance**){  
 **try** {  
 Thread.*sleep*((**int**)(Math.*random*() \* 200));  
 } **catch** (InterruptedException e) {  
 }  
 **balance**-=amount;  
 **return true**;  
 }  
 **return false**;  
 }  
 **public static void** main(String[] args){  
 **final** CheckingAccount ca =**new** CheckingAccount(100);  
 Runnable r = **new** Runnable() {  
 @Override  
 **public void** run() {  
 String name = Thread.*currentThread*().getName();  
 **for**(**int** i=0;i<10;i++){  
 System.***out***.println(name+**"withdraws $10: "**+ca.withdraw(10)+**"amount:"**+ca.**balance**);  
 }  
 }  
 };  
 Thread thdHusband = **new** Thread(r);  
 thdHusband.setName(**"Husband"**);  
 Thread thdWife = **new** Thread(r);  
 thdWife.setName(**"Wife"**);  
 thdHusband.start();  
 thdWife.start();  
 }  
}

运行结果：

两人的共用账户上已经没有了钱，可是Husband线程却仍取出了10美元，数据没有同步，Husband读取的账户余额为脏数据而不是最新的数据，使得系统认为还有余额，便让其取出10美元，写入数据时发现账户余额为负。

1. 修复前面CheckingAccount类的线程问题。

Volatile关键字能解决可见性问题，他在读或写的时候都是直接与内存打交道而不是缓存，避免读取到还未存入内存的新数据并执行后续代码。

Synchronized确保事务的原子性，同时只能有一个线程对balance进行读写。

