1.

多线程有两种创建方式，一种是继承Thread类，一种是实现Runnable接口。推荐使用第二种方法，因为java中不支持多继承。但是可以多实现。

2.以下是继承Thread实现的多线程

**public** **class** ThreadDemo {

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

// **TODO** Auto-generated method stub

A a = **new** A();

a.start();

**for** (**int** i = 0; i < 60; i++) {

System.***out***.println("main run -- " + i);

}

a.~~stop~~();

}

}

**class** A **extends** Thread{

**public** **void** run() {

**for** (**int** i = 0; i < 60; i++) {

System.***out***.println("A run -- " + i);

}

}

}

3.以下是实现Runnable接口实现的多线程

**public** **class** RunnableDemo {

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

A a = **new** A();

Thread t = **new** Thread(a);

t.start();

**for** (**int** i = 0; i < 60; i++) {

System.***out***.println("main -- " + i);

}

}

}

**class** A **implements** Runnable{

@Override

**public** **void** run() {

**for** (**int** i = 0; i < 60; i++) {

System.***out***.println("A -- " + i);

}

}

}

4.

子类复写或实现的run方法不能抛异常。

5.

获取当前线程的名称，使用Thread.currentThread().getName()

6.

同步代码块关键字是synchronized，作用是：保证共享数据某一时间只有一个线程操作，这个线程操作没有结束时，其他线程无法操作此数据。

7.

使用同步的前提：1）必须要有两个或两个以上的线程。2）多个线程必须持同一把锁

8.

非静态同步函数用的是this锁。静态的同步函数用的是类名.class锁。使用示例：public static synchronized void fun(){}

9.以下是一个死锁程序示例：

**public** **class** DeadLockDemo {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

LockA la = **new** LockA();

LockB lb = **new** LockB();

Thread t = **new** Thread(la);

Thread t1 = **new** Thread(lb);

t.start();

t1.start();

Thread.*sleep*(10);

**synchronized** (Lock.*o1*) {//false

System.***out***.println(Thread.*currentThread*().getName() + " :this is B lock");

}

}

}

**class** Lock{

**public** **static** Object *o1* = **new** Object();

**public** **static** Object *o2* = **new** Object();

**public** **static** Object *o3* = **new** Object();

}

**class** LockA **implements** Runnable{

@Override

**public** **void** run() {

**synchronized** (Lock.*o1*) {//false

**try** {Thread.*sleep*(10);}**catch**(Exception e) {}

System.***out***.println(Thread.*currentThread*().getName() + " :this is B lock");

**synchronized** (Lock.*o2*) {//false

show();

}

}

}

**public** **void** show() {

System.***out***.println(Thread.*currentThread*().getName() + " :this is A lock");

}

}

**class** LockB **implements** Runnable{

@Override

**public** **void** run() {

**synchronized** (Lock.*o2*) {//false

**try** {Thread.*sleep*(10);}**catch**(Exception e) {}

System.***out***.println(Thread.*currentThread*().getName() + " :this is B lock");

**synchronized** (Lock.*o1*) {//false

show();

}

}

}

**public** **void** show() {

System.***out***.println(Thread.*currentThread*().getName() + " :this is A lock");

}

}

每一个对象就是一把锁，无论他在哪，这个对象的锁都是一定的，要么开，要么关。如果程序有线程（p1, p2, p3……pn），p1拿到了锁1的钥匙，等着p2吧p2的钥匙给p1，然而p2拿着自己的钥匙，等着p3把p3的钥匙给p2，如此便进入了一个循环，每个人都拿着自己手里的钥匙不放手，但每个人都等着别人拿钥匙过来。于是就出现了死锁现象。

上面的程序示例，t拿到了o1钥匙后，把o1锁上了，此时t1抢到了CPU的执行权，他拿到了o2的钥匙，把o2锁上了。然后t抢到了执行权，但是o2被锁了，他需要等t1执行完后才能进入。然后t1拿到了执行权，但是o1被锁了，他需要等t执行完后才能进入。于是双方都在等待。也就进入了死锁。

10.

notify方法的解释

Wakes up all threads that are waiting on this object's monitor. A

thread waits on an object's monitor by calling one of the

{@code wait} methods.

百度翻译：唤醒正在等待此对象监视器的所有线程。A线程通过调用@code wait方法。

可见，每个锁都有一个线程池队列，要唤醒某线程，就要持有某个线程的锁，访问该锁的线程池，然后唤醒队头线程。

11.

异常：java.lang.IllegalMonitorStateException

解释：if the current thread is not the owner of this object's monitor.

如果当前线程不是此对象监视器的所有者就会抛出此异常。

原因是：在对某个对象上调用wait()方法进行线程等待（让其他竞争执行该代码的线程上锁）时，没有对该对象执行同步操作。  
 故解决办法是：  
    synchronized (xxxx)  {  
      xxxx.wait();

xxxx.notify();  
   }

12.

Join()方法: 用于抢夺CPU执行权。当A线程使用了join方法，其他线程都要等A线程运行完后才能继续运行。

13.

线程的优先级是1~10级，一般线程被创建后，优先级都是5，但是可以使用setPriority(int num)方法设置线程优先级。

14.

Yield方法可以暂停当前线程的运行，让别的线程先运行