Java SE Lesson 15

1. 一个类若想被序列化，则需要实现 java.io.Serializable 接口，该接口中没有定义任何方法，是一个标识性接口（Marker Interface），当一个类实现了该接口，就表示这个类的对象是可以序列化的。
2. 在序列化时，static 变量是无法序列化的；如果 A 包含了对 B 的引用，那么在序列化 A 的时候也会将 B 一并地序列化；如果此时 A 可以序列化，B 无法序列化，那么当序列化 A 的时候就会发生异常，这时就需要将对 B 的引用设为 transient，该关键字表示变量不会被序列化。

```
private void writeObject(java.io.ObjectOutputStream out) throws IOException
{
    out.writeInt(age);
    out.writeUTF(name);

    System.out.println("write object");
}

private void readObject(java.io.ObjectInputStream in) throws IOException,
        ClassNotFoundException
{
    age = in.readInt();
    name = in.readUTF();

    System.out.println("read object");
}
```

3. 当我们在一个待序列化/反序列化的类中实现了以上两个 private 方法（方法声明要与上面的保持完全的一致），那么就允许我们以更加底层、更加细粒度的方式控制序列化/反序列化的过程。
4. Java 中如果我们自己没有产生线程，那么系统就会给我们产生一个线程（主线程，main 方法就在主线程上运行），我们的程序都是由线程来执行的。
5. 进程：执行中的程序（程序是静态的概念，进程是动态的概念）。
6. 线程的实现有两种方式，第一种方式是继承 Thread 类，然后重写 run 方法；第二种是实现 Runnable 接口，然后实现其 run 方法。
7. 将我们希望线程执行的代码放到 run 方法中，然后通过 start 方法来启动线程，**start 方法首先为线程的执行准备好系统资源，然后再去调用 run 方法**。当某个类继承了 Thread 类之后，该类就叫做一个线程类。
8. 一个进程至少要包含一个线程。
9. 对于单核 CPU 来说，某一时刻只能有一个线程在执行（微观串行），从宏观角度来看，多个线程在同时执行（宏观并行）。
10. 对于双核或双核以上的 CPU 来说，可以真正做到微观并行。
11.

1) Thread 类也实现了 Runnable 接口，因此实现了 Runnable 接口中的 run 方法；
2) 当生成一个线程对象时，如果没有为其设定名字，那么线程对象的名字将使用如下形式：Thread-**number,该 number 将是自动增加的，并被所有的 Thread 对象所共享（因为它是 static 的成员变量）。**
3) 当使用第一种方式来生成线程对象时，我们需要重写 run 方法，因为 Thread 类的 run 方法此时什么事情也不做。

4) 当使用第二种方式来生成线程对象时，我们需要实现 Runnable 接口的 run 方法，然后使用 new Thread(new MyThread())（假如 MyThread 已经实现了 Runnable 接口）来生成线程对象，这时的线程对象的 run 方法就会调用 MyThread 类的 run 方法，这样我们自己编写的 run 方法就执行了。

12. 关于成员变量与局部变量：如果一个变量是成员变量，那么多个线程对同一个对象的成员变量进行操作时，他们对该成员变量是彼此影响的（也就是说一个线程对成员变量的改变会影响到另一个线程）。
13. 如果一个变量是局部变量，那么每个线程都会有一个该局部变量的拷贝，一个线程对该局部变量的改变不会影响到其他的线程。
14. 停止线程的方式：不能使用 Thread 类的 stop 方法来终止线程的执行。一般要设定一个变量，在 run 方法中是一个循环，循环每次检查该变量，如果满足条件则继续执行，否则跳出循环，线程结束。
15. 不能依靠线程的优先级来决定线程的执行顺序。
16. synchronized 关键字：当 synchronized 关键字修饰一个方法的时候，该方法叫做同步方法。
17. **Java 中的每个对象都有一个锁（lock）或者叫做监视器（monitor），当访问某个对象的 synchronized 方法时，表示将该对象上锁，此时其他任何线程都无法再去访问该 synchronized 方法了，直到之前的那个线程执行方法完毕后（或者是抛出了异常），那么将该对象的锁释放掉，其他线程才有可能再去访问该 synchronized 方法。**
18. 如果一个对象有多个 synchronized 方法，某一时刻某个线程已经进入到了某个 synchronized 方法，那么在该方法没有执行完毕前，其他线程是无法访问该对象的任何 synchronized 方法的。