复习：

1. 转换流：转换输出流是将字符转化为字节，转换输入流是将字节转化为字符 ---最底层都是以字节形式来读取或者写出数据，表现形式都是以字符形式来展现---转换流都是字符流

2. 系统流：System.in，System.out，System.err---都是字节流---系统流本质上是静态对象---一旦关闭就无法使用---系统流在使用的时候一般不用关闭

3. 打印流：PrintStream, PrintWriter---允许便捷的打印数据

4. 序列化/反序列化

序列化：将对象的信息进行完整保存 --- 持久化

一个接口---Serializable，两个关键字---static/transient，版本号--- serialVersionUID --- 保证在类产生微小变动的情况下已经序列化的出去的对象能够反序列化回来，集合/Map---元素不能随着集合一起序列化出去

5. 静态导入：import static 包名.类名.方法名 --- 表示导入某个已有类中的指定的静态方法

6. 可变参数：用 ... 来定义，本质上是一个数组。参数的个数不做限定。可变参数只能定义一个而且必须放在参数列表的末尾

7. 枚举：用enum来定义枚举。枚举常量必须放在第一行。枚举中的构造方法默认是私有的。枚举可以定义方法和属性，包括抽象方法。枚举的父类是Enum。从jdk1.5开始，允许在switch-case中使用枚举常量

8. 单元测试：需要单独导入测试库---Junit4，在要测试的方法上加上@Test，要求测试的方法不能有参数，无返回值，是一个非静态方法。@Before---一般用于完成一些初始化的操作，@After---用于善后

## 线程

进程：计算机中执行的任务---服务一般没有界面

线程：进程中的一个小任务 QQ/微信、下载软件、记事本、JVM。

注意：计算机的每一个核在每一个时刻都只能处理一个进程中的一个线程---宏观上是并行，微观上是串行

扩展：Windows中默认是用一个核处理，Linux下是有几个核就用几个核

### 定义线程

1. 继承Thread，重写run方法，将逻辑放到run方法中，开启线程用的是start方法

2. 实现Runnable，重写run方法，需要利用Thread对象来启动这个线程

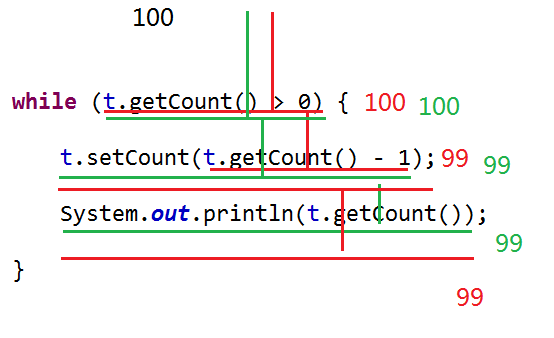
3. 实现Callable<T>，重写call方法 --- 现阶段知道即可

### 多线程的并发安全问题

因为多个线程之间相互抢占而导致出现不符合常理的现象就称之为多线程的并发安全问题

线程的执行是相互抢占资源的。---谁抢到谁就执行。

线程的抢占是发生在线程执行的每一步过程中。



同步代码块---synchronized---意味着一个线程在执行了其他线程不能执行---需要锁对象---锁对象要求被所有的线程都认识：共享资源、类的字节码、this

同步：一段逻辑在一个时间段内只允许一个线程执行

异步：一段逻辑在一个时间段内只允许多个线程执行

同步一定安全---是

不安全一定异步---是

安全不一定同步---宏观角度---微观上安全一定同步

### 死锁

由于锁导致多个线程相互等着致使程序无法继续往下执行

导致的原因：多线程、较多的共享资源、锁的嵌套

避免死锁的方法：减少线程数量、减少共享资源、统一锁对象、避免锁嵌套

### 等待唤醒机制

利用的wait和notify来控制线程的执行顺序---等待唤醒机制

练习：生产消费模型

一个线程表示生产者，另一个线程表示消费者。生产一次然后消费一次

生产者在生产的时候要求本次生产的商品数量和上次剩余的商品数量的和不能超过1000，本次的消费数量不超过提供的商品数量

0 500 500

500 420 80

80 800 880

880 10 870

870 130 1000

1000 0 1000

1000 0 1000

线程在等待的时候是去线程池中等待的。线程池本质上是一个存储线程的队列。---遵循先进先出的原则

如果出现了多个线程，只唤醒一个线程就不够用了，所以用notifyAll唤醒所有的线程

注意：等待唤醒机制必须结合锁对象来使用---等待唤醒和锁是对应的

总结：sleep和wait有什么区别？

sleep需要休眠时间，到点自然醒。会释放执行权，没有释放锁。设计在了Thread类上，是一个静态方法。

wait可以不指定等待时间，也可以指定等待时间。如果没有指定等待时间需要手动唤醒。wait必须结合锁使用。释放执行权，释放锁。设计在了Object上，是一个成员方法。

### 守护线程

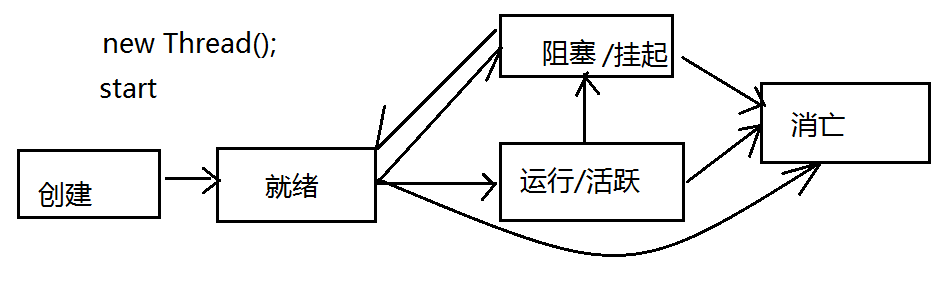
守护其他线程，被守护的线程结束的时候，守护线程无论完成与否都要结束

一个线程要么是守护线程要么是被守护的线程---当最后一个被守护的线程结束才会导致所有的守护线程结束---GC

### 线程的优先级

优先级一共有1-10这十个等级。理论上数字越大优先级越高，也就意味着线程抢到资源的几率就越大。实际中相邻两个优先级的差别很不明显。至少相差5个优先级及以上才会相对明显

### 线程的状态



考虑：引入多线程模型的意义在哪儿？

减少响应时间，提高CPU的利用率

1 0.2 0.8

2 0.36 0.8\*0.8=0.64

3 0.8\*0.8\*0.8

## 单例模式

在全局过程中只有一个实例。

对象定义的时候就初始化叫饿汉式

对象在真正使用的时候才初始化叫懒汉式

饿汉式会增加加载过程，懒汉式能够避免这个问题。

懒汉式会出现线程安全问题，饿汉式不会出现线程安全问题

扩展：掌握单例模式的七种方式