**关于类：**

1．不可以单独一个类是private，其他的类如果你这样写了哪么就意味着外部不能调用他除了他自己调用之外，这样的类是没有意义的。所以除了[内部类](https://www.baidu.com/s?wd=%E5%86%85%E9%83%A8%E7%B1%BB&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)之外是不能用private修饰类的！

**关于异常：**

1.在写程序时，对可能会出现异常的部分通常要用try{...}catch{...}去捕捉它并对它进行处理；

2.用try{...}catch{...}捕捉了异常之后一定要对在catch{...}中对其进行处理，那怕是最简单的一句输出语句，或栈输入e.printStackTrace();

3.如果是捕捉IO输入输出流中的异常，一定要在try{...}catch{...}后加finally{...}把输入输出流关闭；

4.如果在函数体内用throw抛出了某种异常，最好要在函数名中加throws抛异常声明，然后交给调用它的上层函数进行处理。

5.需要在本级处理的就try catch，需要交给上级处理的就throws

**关于抽象类和接口：**

6.抽象类 public abstract class 名称{里面要有抽象方法},抽象类的构造方法定义为protected。

个人理解，抽象类的好处可以用于继承，父类有a，b,c继承了a, 然后a指向b,c，可以直接用父类方法实现多态（直接一个父类搞定，当然也可以在子类里分别写，用不同的子类进行比较）。

7.抽象方法不能包含在非抽象类中，抽象类不能用new创建实例，，但是可以使用new创建抽象类数组。

8.接口只能包含常量和抽象方法，用于描述多个对象的共同行为。Public interface 名称{};使用接口是为了让不属于一个家族的类都可以使用相同的方法。

9.接口中所有数据都是public final static，所有的方法都是public abatract，所以系统默认可以不写。接口没有构造方法，不能用new,所有方法必须公共的。

**关于泛型：**

1. 泛型指的是参数化类型，即类型在初始化时可以变化的。
2. 泛型可以分为泛型类、泛型接口、泛型方法。
3. 泛型类:public class 名称<T>，用到类型T时返回即可。
4. 泛型方法比较复杂
5. 泛型接口：public interface 名称<T>{}

**集合框架：**

1. Java支持三种主要类型的集合：规则集Set(用于存储一组不重复的数)，线性表list，队列queue。Collection是其基本类。
2. 关于规则集set可以分为散列集合HashSet，链式散列集LinkedHashSet,树形集TreeSet。

关于HashSet（输出顺序不保证是插入顺序）:

1. HashSet使用：Set<T> s =new HashSet<T>，默认初始容量16客座率0.75，及当元素尺度达到12，容量会翻倍到32。
2. 遍历Hashset。法一 Iterator<T> iter=set.iterator();

While(iter.hasNext()){syso(iter,next)}

法二 for(Object o : s){syso(o)}

关于LinkedHashSet（输出顺序保证是插入顺序）:

Set<T> s=new LinkedHashSet<T>();

关于树形集TreeSet:

a)TreeSet 是二差树实现的,Treeset中的数据是自动排好序的，不允许放入null值（HashSet 可以有一个NUll）。

课后看 ArrayList LinkedList Vector Stack 队列等

Hash Map的使用：

Map(T,T) h =new HashMap(T,T)类似于字典

Put(key,value), get(key)

**关于NIO（未完待续）：**

1. Java NIO和IO之间第一个最大的区别是，IO是面向流的（每次从流中读取字节没有被缓存在任何地方，不能前后移动流中数据），NIO是面向缓冲区的（把数据读取到缓冲区，可以在缓冲区前后移动，灵活）。

多线程：

1.使用多线程包含两种方法（lang包）：

法1：使用Runnable接口重写run方法

思路：

1. 编写自己的任务类（包含构造函数等）并implements Runnable接口，在自己的任务类中重写Runnable接口中的run方法（该方法返回值为void）
2. 在主函数中先 new出自己的任务类(父类可是自己的任务类，或者Runnable类)，调用Thread类并把任务类导入（Thread t = new Thread（任务类的实例）），启动线程（t.start()）

法2： 任务类继承Thread类，重写run方法

1. 写好自己的任务类并继承Thread类（task extends Thread），接着重写run方法；
2. 在主函数中new出自己的任务类(父类是自己的任务类，不可以是Runnable)，接着不用像法1那样定义Thread类的实例了，直接start自己写的任务类实例即可。
3. Thread.yield()//结束该次的线程，cpu让出
4. 如果同时有两个线程，线程A(主线程)和线程B。如果你不知道线程B需要执行多长时间，并且在A中需要使用到B中的结果，考虑使用join方法。例：

MyThread myThread = new MyThread();

myThread.start();

myThread.join();

System.out.println("Mythread执行完了之后再执行这里");

这样写能够保证syso是在myThread执行完后执行的，否则会乱窜。可以把join理解为该实例跑完了才可以走后面。

重载方法：A.join（1000），该句话意思是A会占用1s,往后开始竞争占用cpu。

1. 停止线程：
2. 设置标志，while(没有达到条件){do something}。
3. Stop 强制停止。 实例.stop()//这样做很危险。
4. While(!this.isInterrupted()){}//没结束时dosomething
5. 关于线程池
6. 动机：如果并发的线程数量很多，并且每个线程都是执行一个时间很短的任务就结束了，这样频繁创建线程（前面的方法）就会大大降低系统的效率，因为频繁创建线程和销毁线程需要时间。
7. 创建线程池的方法(这些方法都不用new)：
8. 线程池的大小无限大，线程池中的线程空闲时间超过1min会被收回。

ExecutorService eService= Executors.newCachedThreadPool();

1. 创建固定大小的可复用的线程池

ExecutorService eS= Executors.newFixedThreadPool(5);

1. 创建延后定期执行的线程池

ScheduledExecutorService sT = Executors.newScheduledThreadPool(5);

1. 创建单线程池，以队列的形式先进先出执行

ExecutorService sT = Executors.newSingleThreadExecutor();

的

1. 创建线程池后：

sT.execute(new 自己的任务类)

执行完毕后sT.shutdown().

//与单个任务相比，不用在Thread一个实例了。

# 关于线程同步---实现互斥（synchronous ）

1. 实景：假设有一个账户，100个线程都往里放1元钱，那么结果是不可能有100元钱，因为会线程可能会同时往里放钱产生覆盖---存在竞争状态，如果线程中没有导致竞争状态，那么线程就是安全的。
2. 同步锁：一个线程进行前加锁，用完之后解锁其他线程才能使用。
3. 同步锁一般用try{}catch{}finally{}模块。使用方法:
4. Lock l =new ReentrantLock();（静态变量加静态锁）
5. 对所在需要同步锁的情节使用l.lock（）
6. 在finally中关闭锁：l.unlock（）
7. 同步互斥锁表明一个时刻只有一个资源被线程调度，为了更灵活使用线程，考虑线程间的合作。
8. Condition c = lock.newCondition();//Lock在上述方法中实例化
9. c.await()//此时后面的线程都需要等待
10. c.signal（）//唤醒一个等待线程
11. c.signalAll（）//唤醒所有等待线程

**关于注解**

[1. 几种常用的注解： a)@Deprecated](mailto:1.@Deprecated)声明这个方法是不建议使用的

B）Override 对父类重写

c)SuppressWarnings（{“unchecks“,”rawtypes”}）//定义泛型时压制泛型警告

2. 自定义注解

a)Annotation是一个注解的根类接口，

b)public @interface 自定义的注解名

未完待续