**1.接口和抽象类区别？接口可以有实现的方法吗/**

**答：1.8以前**

**I.接口只能定义全局常量+抽象方法默认修饰符是public**

**抽象类可以定义抽象方法+普通方法除了抽象方法使用public修饰符外其他方法可以使用任意修饰符，**

**II.子类使用：接口用的是implements关键字 ，抽象类使用extends关键字**

**III.关系：接口不能继承抽象类，但是可以继承多个接口，抽象类可以实现多个接口**

**IV.子类限制：一个子类只能继承一个抽象类，一个子类可以实现多尔接口**

**Jdk1.8以后接口可以有实现的方法体，必须用静态方法**

**2.谈谈String,StringBuffer,StringBuilder的区别？**

**答：String对象是不可变的，底层通过一个不可变的字符数组来保存字符串，StringBuffer和StringBuilder都继承子AbstractStringBuilder类，在AbstractStringBuilder中也是用一个字符数据接收字符串，但是没有用final修饰，所以这两种对象是可变的**

**线程安全这方面：由于String类用final修饰，所以他是线程安全的，StringBuffer通过查看源码，它的方法都是synchronized修饰，所以也是线程安全的，StringBuilder是线程不安全的，**

**3.Java中多线程一共有几种实现方式，你最常用哪一种，为什么？**

**答：有四种方式：**

1. **继承Thread类**
2. **实现Runable接口**
3. **实现Callable**
4. **线程池**

**我最常用的是线程池，**

1. **降低资源消耗。通过重复利用已经创建好的线程降低线程的创建和销毁造成的消耗**
2. **提高响应速度。当任务到达时，如果有空闲线程，任务可以不需要等待线程创建就可以立即执行**
3. **提高线程可管理性。线程是稀缺资源，如果无限制的创建会消耗系统资源，还会降低系统的稳定性，使用线程池可以统一分配，调优和监控**

**延伸：既然说到线程池，那么你说说线程池的创建以及里面的参数和工作流程**

**答：线程池的创建：**

1. **通过构造方法实现**

**A)ThreadPoolExecutor(int,int,TimeUnit,BlockQueue<Runable>)**

**B)ThreadPoolExecutor(int,int,TimeUnit,BlockQueue<Runable>,ThreadFactory)**

**C)ThreadPoolExecutor(int,int,long,TimeUnit,BlockQueue<Runable>,RejectedExceptionHandler)**

1. **通过Executor框架的工具类Executors来实现，我们可以创建三种类型的ThreadPoolExecutor:**
   1. **newFixedThreadPool(int):返回固定的线程数量的线程池**
   2. **newSingleThreadExecutor:返回只有一个线程的线程池**
   3. **newCachedThreadPool:返回一个可根据实际情况调整线程数量的线程池，线程数量不稳定，当一个任务提交到线程池，如果没有空闲线程数，则会创建，若有空线程，则可以复用，**

**线程池参数有：**

1. **核心线程数**
2. **最大线程数**
3. **线程存活时间**
4. **时间单位**
5. **阻塞队列**
6. **拒绝策略(默认抛出异常)**

**工作流程：**

1. **当一个任务提交到线程池中，首先判断核心线程池的线程数是否有空闲,有的话直接执行，否则进行第二步**
2. **判断阻塞队列是否已满，如果没有满的话，加入阻塞队列，否则执行第三步**
3. **判断非核心线程池的线程数是否有空闲，如果有，执行该任务，否则执行第四步**
4. **调用相应的拒绝策略打回任务，默认抛出一个异常**

**知道设计模式吗？比如单例设计模式，工厂设计模式，手写一个单例设计模式**

**public class Singleton{**

**private static Singleton singleton;**

**private Singleton(){}**

**public static Singleton getInfo(){**

**if(singlenton == null){**

**synchronized(Singleton.class){**

**if(singleton == null){**

**singleton = new Singleton();**

**}**

**}**

**}**

**return singleton;**

**}**

**}**

**为什么重写equals就要重写hashcode呢？**

**首先说一下，为什么要重写equals和hashcode？我们先定义一个User类，属性就是name和age，我们一般都是在HashMap，HashSet这种集合存放对象元素时，会重写对象的equals和hashcode方法，我们都知道HashMap存放的键不能重复，那么它是如何保证数据不重复呢？它是通过调用键对象的hashcode和equals来判断对象是否相同，如果对象没有重写hashcode和equals，它就会调用父类Object的hashcode和equals方法，那么当我们new 两个对象时user1,user2，并且使他们的属性都相等。就会分配两块内存地址，此时对应一个hashcode，那么他们的hashcode和equals当然不相同，调用put方法时，就会将两个对象都加入map集合中，这就违背了hashMap键的唯一性**

**下面再来说为什么重写了equals还要重写hashcode，**

**还是上面的例子，现在我们重写了equals方法，里面判断条件是name和age属性相同我们就认为是同一对象，当我们再次调用put方法时，再输出map集合的大小时，发现大小是2，我们都重写了equals，两个对象都相等了，为什么两个都加入map集合中，因为调用put方法时，我们同时判断该对象的hashcode是否和其他对象相同，因为我们没有重写hashcode，所以调用是父类Object的hashcode，他们当然不相同，所以就认为他们不是同一对象，那么map集合大小是2也就解释通了，**

**所以当我们重写equals就要重写hashcode!!!**

**讲一讲你所知道Object类里面的方法，分别是做什么的**

1. **Class<T> getClass() 返回该对象的class对象**
2. **equals()方法 判断和其他对象是否相等**
3. **hashcode() 返回对象的散列码值**
4. **wait() 当前线程等待**
5. **wait(long timeout,int nanos )**
6. **wait(long time)**
7. **notify() 唤醒一个等待线程**
8. **notifyAll() 唤醒所有等**
9. **待线程**
10. **finalize() gc时会调用**
11. **Object clone() 返回一个拷贝的object对象**
12. **toString()**
13. **1.会涉及反射，双亲委派机制，类加载机制。**
14. **2,3会扯到HashMap和HashSet上面**
15. **4,5,6,7,8 会扯到多线程**
16. **9 会扯到垃圾回收机制(垃圾回收算法，如何判断对象被回收...)**

**反射相关：**

**Synchronized 和 ReenTrantLock区别**

1. **Synchronized和ReenTrantLock都是可重入锁**
2. **相比较**S**ynchronized，ReenTrantLock实现了等待可中断(lockInterruptibly)，公平锁(根据构造方法传递参数实现)，选择性通知**
3. **Synchronized和wait，notify，notifyAll实现等待通知机制，并且只有一个等待队列，而ReenTrantLock也可以实现等待通知机制，它需要与Condition相结合使用，并且它有多个Condition队列**

**Synchronized和volatile区别**

1. **Synchronized可以修饰方法，代码块，volatile可以修饰变量**
2. **Volatile只能保证数据的可见性不能保证原子性，所以主用于解决变量在多个线程之间的可见性，而Synchronized两者都可以保证，它解决的是多个线程访问资源的同步性**
3. **volatile不会发生阻塞，Synchronized可能发生阻塞**