**1.抽象和封装的不同点**

抽象和封装是互补的概念。一方面，抽象关注对象的行为。另一方面，封装关注对象行为的细节。一般是通过隐藏对象内部状态信息做到封装，因此，封装可以看成是用来提供抽象的一种策略。

**2.重载和重写的区别**

重载： 发生在同一个类中，方法名必须相同，参数类型不同、个数不同、顺序不同，方法返回值和访问修饰符可以不同，发生在编译时。

重写： 发生在父子类中，方法名、参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类，访问修饰符范围大于等于父类；如果父类方法访问修饰符为private则子类就不能重写该方法。

**3.字符型常量和字符串常量的区别**

字符常量是单引号引起的一个字符 字符串常量是双引号引起的若干个字符型常量相当于一个整形值(ASCII值),可以参加表达式运算 字符串常量代表一个地址值(该字符串在内存中存放位置)字符常量只占一个字节 字符串常量占若干个字节(至少一个字符结束标志)

**4.成员变量与局部变量的区别有那些？**

从语法形式上，看成员变量是属于类的，而局部变量是在方法中定义的变量或是方法的参数；成员变量可以被public,private,static等修饰符所修饰，而局部变量不能被访问控制修饰符及static所修饰；但是，成员变量和局部变量都能被final所修饰；从变量在内存中的存储方式来看，成员变量是对象的一部分，而对象存在于堆内存，局部变量存在于栈内存从变量在内存中的生存时间上看，成员变量是对象的一部分，它随着对象的创建而存在，而局部变量随着方法的调用而自动消失。成员变量如果没有被赋初值，则会自动以类型的默认值而赋值（一种情况例外被final修饰但没有被static修饰的成员变量必须显示地赋值）；而局部变量则不会自动赋值。

**5.讲讲对static的理解？Java中是否可以覆盖一个private或者是static的方法？**

如果一个类的变量或者方法前面有static修饰，那么表明这个方法或者变量属于这个类，也就是说可以在不创建对象的情况下直接使用

当父类的方法被private修饰时，表明该方法为父类私有，对其他任何类都是不可见的，因此如果子类定了一个与父类一样的方法，这对于子类来说相当于是一个新的私有方法，且如果要进行向上转型，然后去调用该“覆盖方法”，会产生编译错误

static方法时编译时静态绑定的，属于类，而覆盖是运行时动态绑定的(动态绑定的多态),因此不能覆盖.

**6.是否可以在static环境中访问非static变量？**

static变量在Java中是属于类的，它在所有的实例中的值是一样的。

当类被Java虚拟机载入的时候，会对static变量进行初始化。

如果代码尝试不用实例来访问非static的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

**7.Java支持的基本数据类型有哪些？**

java支持的基本数据类型有以下8种:byte,short,int,long,float,double,char,boolean.

**8.怎么理解JAVA的自动拆箱装箱？**

所谓自动装箱就是将基本数据类型自动的转换为对应的对象包装类型，而拆箱就是将对象包装类型转换为基本数据类型。

java中的自动拆装箱通常发生在变量赋值的过程中，如：把int转化成Integer，double转化成double就是**自动装箱**，反之就是**自动拆箱**

在实际中，应该注意自动拆装箱，因为有时可能因为java自动装箱机制，而导致创建了许多对象，对于内存小的平台会造成压力。

**9. 重写和重载是什么?**

重写：发生在子类与父类之间，表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。重载：是指在一个类中，可以有多个相同名称的方法，但是他们的参数列表的个数或类型不同，当调用该方法时，根据传递的参数类型调用对应参数列表的方法。当参数列表相同但返回值不同时，将会出现编译错误，这并不是重载，因为jvm无法根据返回值类型来判断应该调用哪个方法。

**10.Java支持多继承么？如果不支持，如何实现?**

不支持，Java不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

在java中是单继承的，也就是说一个类只能继承一个父类。

java中实现多继承有两种方式,一是接口，而是内部类.

**11.什么是值传递和引用传递？Java中是值传递还是引用传递，还是都有?**

值传递：就是在方法调用的时候，实参是将自己的一份拷贝赋给形参，在方法内，对该参数值的修改不影响原来实参。引用传递：是在方法调用的时候，实参将自己的地址传递给形参，此时方法内对该参数值的改变，就是对该实参的实际操作。在java中只有一种传递方式，那就是值传递.可能比较让人迷惑的就是java中的对象传递时，对形参的改变依然会影响到该对象的内容。

**12.接口和抽象类的区别是什么?**

接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。类可以实现很多个接口，但是只能继承一个抽象类,类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。抽象类可以在不提供接口方法实现的情况下实现接口。Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public 。接口是绝对抽象的，不可以被实例化(java 8已支持在接口中实现默认的方法)。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的。

**13.构造器（constructor）是否可被重写（override）?**

构造方法是不能被子类重写的，但是构造方法可以重载

简单的讲，就是说一个类可以有多个构造方法。

**14. String, StringBuffer StringBuilder的区别**

String 的长度是不可变的；StringBuffer的长度是可变的，线程安全；如果对一个字符串要经常改变的话，就一定不要用String,否则会创建许多无用的对象出来.

**15.HashMap的工作原理是什么?**

HashMap内部是通过一个数组实现的，只是这个数组比较特殊，数组里存储的元素是一个Entry实体(在JAVA8中为Node)，这个Entry实体主要包含key、value以及一个指向自身的next指针。

HashMap是基于hashing实现的，当进行put操作时，根据传递的key值得到它的hashcode，然后再用这个hashcode与数组的长度进行模运算，得到一个int值，就是Entry要存储在数组的位置（下标）；当通过get方法获取指定key的值时，会根据这个key算出它的hash值（数组下标），根据这个hash值获取数组下标对应的Entry，然后判断Entry里的key，hash值或者通过equals()比较是否与要查找的相同，如果相同，返回value，否则的话，遍历该链表（有可能就只有一个Entry，此时直接返回null），直到找到为止，否则返回null。

HashMap之所以在每个数组元素存储的是一个链表，是为了解决hash冲突问题，当两个对象的hash值相等时，那么一个位置肯定是放不下两个值的，于是hashmap采用链表来解决这种冲突，hash值相等的两个元素会形成一个链表。

**16.HashMap与Hashtable的区别是什么?**

HashMap和Hashtable都实现了Map接口，Hashtable基于Dictionary类，而HashMap是基于AbstractMap类。Dictionary是任何可将键映射到相应值的类的抽象父类，而AbstractMap是基于Map接口的实现，它以最大限度地减少实现此接口所需的工作。HashMap允许键和值是null，而Hashtable不允许键或者值是null。Hashtable是同步(线程安全)的，而HashMap不是同步(非线程安全)。因此，HashMap更适合于单线程环境，而Hashtable适合于多线程环境。HashMap提供了可供应用迭代的键的集合，因此，HashMap是快速失败的。另一方面，Hashtable提供了对键的列举(Enumeration)。

**17.CorrentHashMap的工作原理**

ConcurrenHashMap说是HashMap的升级版

ConcurrentHashMap是线程安全的，但是与Hashtable相比，实现线程安全的方式不同。

Hashtable是通过对hash表结构进行锁定，是阻塞式的，当一个线程占有这个锁时，其他线程必须阻塞等待其释放锁。

ConcurrentHashMap是采用分离锁的方式，它并没有对整个hash表进行锁定，而是局部锁定，也就是说当一个线程占有这个局部锁时，不影响其他线程对hash表其他地方的访问。

ConcurrentHashMap内部有一个Segment<K,V>数组,该Segment对象可以充当锁。Segment对象内部有一个HashEntry<K,V>数组，于是每个Segment可以守护若干个桶(HashEntry),每个桶又有可能是一个HashEntry连接起来的链表，存储发生碰撞的元素。

每个ConcurrentHashMap在默认并发级下会创建包含16个Segment对象的数组，每个数组有若干个桶，当进行put方法时，通过hash方法对key进行计算，得到hash值，找到对应的segment，然后对该segment进行加锁，然后调用segment的put方法进行存储操作，此时其他线程就不能访问当前的segment，但可以访问其他的segment对象，不会发生阻塞等待。

在Java8中，ConcurrentHashMap不再使用Segment分离锁，而是采用一种乐观锁CAS算法来实现同步问题，但其底层还是“数组+链表->红黑树”的实现。

**18.Array和ArrayList有什么区别？**

Array可以包含基本数据类型和对象，而ArrayList只能包含对象。Array大小是固定的，ArrayList的大小是动态变化的。

**19.ArrayList和LinkedList有什么区别？**

ArrayList和LinkedList都实现了List接口，ArrayList是基于数组实现，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。LinkedList是基于链表实现，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。ArrayList在查找时速度快，LinkedList的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。LinkedList比ArrayList更占内存，因为LinkedList为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

**20.哪些集合类提供对元素的随机访问？**

ArrayList、HashMap、TreeMap和HashTable类提供对元素的随机访问。

**21.HashSet的底层实现是什么?**

HashSet的实现是依赖于HashMap的，HashSet的值都是存储在HashMap中的。

在HashSet的构造法中会初始化一个HashMap对象，HashSet不允许值重复。

因此，HashSet的值是作为HashMap的key存储在HashMap中的，当存储的值已经存在时返回false。

**22.Comparable和Comparator接口的区别。**

Comparable接口只包含一个compareTo()方法。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。Comparator接口包含compare()和equals()两个方法。compare()方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。

equals()方法需要一个对象作为参数，它用来决定输入参数是否和comparator相等。只有当输入参数也是一个comparator并且输入参数和当前comparator的排序结果是相同的时候，这个方法才返回true。

**23.HashSet和TreeSet有什么区别？**

HashSet是由一个hash表来实现的，因此，它的元素是无序的。add()，remove()，contains()方法的时间复杂度是O(1)。TreeSet是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains()方法的时间复杂度是O(logn)。

**24. Java中==与equals的区别**

"==" 的作用是判断两个对象的地址是不是相等。即判断两个对象是不是同一个对象。(基本数据类型==比较的是值，引用数据类型==比较的是内存地址)equals() : 它的作用也是判断两个对象是否相等。但它一般有两种使用情况：类没有覆盖equals()方法。则通过equals()比较该类的两个对象时，等价于通过“==”比较这两个对象。类覆盖了equals()方法。一般，我们都覆盖equals()方法来两个对象的内容相等；若它们的内容相等，则返回true(即，认为这两个对象相等)。

**25.你重写过 hashcode 和 equals 么，为什么重写equals时必须重写hashCode方法？**

hashCode() 的作用是获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。如果两个对象相等，则hashcode一定也是相同的如果两个对象相等,对两个对象分别调用equals方法都返回true如果两个对象有相同的hashcode值，它们也不一定是相等的因此，equals方法被覆盖过，则hashCode方法也必须被覆盖

hashCode()的默认行为是对堆上的对象产生独特值。如果没有重写hashCode()，则该class的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

**26.Java的四种引用，强弱软虚，用到的场景**

**强引用**：如果一个对象具有强引用，它就不会被垃圾回收器回收。即使当前内存空间不足，JVM也不会回收它，而是抛出 OutOfMemoryError 错误，使程序异常终止。如果想中断强引用和某个对象之间的关联，可以显式地将引用赋值为null，这样一来的话，JVM在合适的时间就会回收该对象**软引用**：在使用软引用时，如果内存的空间足够，软引用就能继续被使用，而不会被垃圾回收器回收，只有在内存不足时，软引用才会被垃圾回收器回收。**弱引用**：具有弱引用的对象拥有的生命周期更短暂。因为当 JVM 进行垃圾回收，一旦发现弱引用对象，无论当前内存空间是否充足，都会将弱引用回收。不过由于垃圾回收器是一个优先级较低的线程，所以并不一定能迅速发现弱引用对象**虚引用**：顾名思义，就是形同虚设，如果一个对象仅持有虚引用，那么它相当于没有引用，在任何时候都可能被垃圾回收器回收。