含有abstract修饰符的class即为抽象类，abstract类不能创建的实例对象。含有abstract方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为abstract类型。

接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final。
下面比较一下两者的语法区别：

- 抽象类可以有构造方法，接口中不能有构造方法
- 抽象类中可以有普通成员变量，接口中没有普通成员变量
- 抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法
- 抽象类中的抽象方法的访问类型可以是public，protected和（默认类型,虽然eclipse下不报错，但应该也不行），但接口中的抽象方法只能是public类型的，并且默认即为public abstract类型
- 抽象类中可以包含静态方法，接口中不能包含静态方法
- 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是public static final类型，并且默认即为public static final类型
- 一个类可以实现多个接口，但只能继承一个抽象类

如果两个操作数其中有一个是double类型，另一个操作就会转换为double类型

否则，如果其中一个操作数是float类型，另一个将会转换为float类型

否则，如果其中一个操作数是long类型，另一个会转换为long类型

否则，两个操作数都转换为int类型

用new创建的对象在堆区

函数中的临时变量在栈区

java中的字符串在字符串常量区

---

public Method[] getDeclaredMethods()返回类或接口声明的所有方法，包括public, protected, default (package) 访问和private方法的Method对象，但不包括继承的方法。当然也包括它所实现接口的方法。

public Method[] getMethods()返回类的所有public方法，包括其继承类的公用方法，当然也包括它所实现接口的方法。

包装类的“==”运算在不遇到算术运算的情况下不会自动拆箱

包装类的equals()方法不处理数据转型

ThreadLocal的类声明：public class ThreadLocal 可以看出ThreadLocal并没有继承自Thread，也没有实现Runnable接口

ThreadLocal类为每一个线程都维护了自己独有的变量拷贝。每个线程都拥有了自己独立的一个变量。所以ThreadLocal重要作用并不在于多线程间的数据共享，而是数据的独立

由于每个线程在访问该变量时，读取和修改的，都是自己独有的那一份变量拷贝，不会被其他线程访问，变量被彻底封闭在每个访问的线程中

ThreadLocal中定义了一个哈希表用于为每个线程都提供一个变量的副本

Java中的String是一个类，而并非基本数据类型。string是值传入，不是引用传入。

StringBuffer和StringBuilder可以算是双胞胎了，这两者的方法没有很大区别。但在线程安全性方面，StringBuffer允许多线程进行字符操作。 这是因为在源代码中StringBuffer的很多方法都被关键字 synchronized 修饰了，而StringBuilder没有。

StringBuilder的效率比StringBuffer稍高，如果不考虑线程安全，StringBuilder应该是首选。另外，JVM运行程序主要的时间耗费是在创建对象和回收对象上。

String对String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，而不是StringBuffer；StringBuffer每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。

---

final修饰类、方法、属性！不能修饰抽象类，因为抽象类一般都是需要被继承的，final修饰后就不能继承了。

final修饰的方法不能被重写而不是重载！

final修饰属性，此属性就是一个常量，不能被再次赋值！

简单记忆线程安全的集合类： 喂！SHE！ 喂是指 vector，S是指 stack， H是指 hashtable，E是指：Eenumeration

---

重载：函数方法名必须相同，看参数列表即可，无关返回值

重载的依据是参数列表不同，返回值不能作为重载的依据

---

throw用于抛出异常。

throws关键字可以在方法上声明该方法要抛出的异常，然后在方法内部通过throw抛出异常对象。

try是用于检测被包住的语句块是否出现异常，如果有异常，则抛出异常，并执行catch语句。

cacth用于捕获从try中抛出的异常并作出处理。

finally语句块是不管有没有出现异常都要执行的内容。

泛型仅仅是java的语法糖，它不会影响java虚拟机生成的汇编代码，在编译阶段，虚拟机就会把泛型的类型擦除，还原成没有泛型的代码，顶多编译速度稍微慢一些，执行速度是完全没有什么区别的

成员变量有初始值，而局部变量没有初始值得。变量没有初始值就使用了，编译通不过

解决哈希冲突常用的两种方法是：开放定址法和链地址法

开放定址法：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。沿此序列逐个单元地查找，直到找到给定 的关键字，或者碰到一个开放的地址(即该地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。查找时探查到开放的 地址则表明表中无待查的关键字，即查找失败。

链地址法：将所有关键字为同义词的结点链接在同一个单链表中。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数 组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。

---

bootstrap classloader －引导（也称为原始）类加载器，它负责加载Java的核心类。

extension classloader －扩展类加载器，它负责加载JRE的扩展目录（JAVA_HOME/jre/lib/ext或者由java.ext.dirs系统属性指定的）中JAR的类包。

system classloader －系统（也称为应用）类加载器，它负责在JVM被启动时，加载来自在命令java中的-classpath或者java.class.path系统属性或者CLASSPATH*作系统属性所指定的JAR类包和类路径。

1.从地址栏显示来说

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址. redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.

2.从数据共享来说

forward:转发页面和转发到的页面可以共享request里面的数据. redirect:不能共享数据.

3.从运用地方来说

forward:一般用于用户登陆的时候,根据角色转发到相应的模块. redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等.

4.从效率来说

forward:高. redirect:低.

本质区别

解释一

一句话，转发是服务器行为，重定向是客户端行为。为什么这样说呢，这就要看两个动作的工作流程： 转发过程：客户浏览器发送http请求----》web服务器接受此请求--》调用内部的一个方法在容器内部完成请求处理和转发动作----》将目标资源 发送给客户；在这里，转发的路径必须是同一个web容器下的url，其不能转向到其他的web路径上去，中间传递的是自己的容器内的request。在客 户浏览器路径栏显示的仍然是其第一次访问的路径，也就是说客户是感觉不到服务器做了转发的。转发行为是浏览器只做了一次访问请求。

重定向过程：客户浏览器发送http请求----》web服务器接受后发送302状态码响应及对应新的location给客户浏览器--》客户浏览器发现 是302响应，则自动再发送一个新的http请求，请求url是新的location地址----》服务器根据此请求寻找资源并发送给客户。在这里 location可以重定向到任意URL，既然是浏览器重新发出了请求，则就没有什么request传递的概念了。在客户浏览器路径栏显示的是其重定向的 路径，客户可以观察到地址的变化的。重定向行为是浏览器做了至少两次的访问请求的。

解释二

重定向，其实是两次request, 第一次，客户端request A,服务器响应，并response回来，告诉浏览器，你应该去B。这个时候IE可以看到地址变了，而且历史的回退按钮也亮了。重定向可以访问自己web应用以外的资源。在重定向的过程中，传输的信息会被丢失。

解释三

假设你去办理某个执照， 重定向：你先去了A局，A局的人说：“这个事情不归我们管，去B局”，然后，你就从A退了出来，自己乘车去了B局。

转发：你先去了A局，A局看了以后，知道这个事情其实应该B局来管，但是他没有把你退回来，而是让你坐一会儿，自己到后面办公室联系了B的人，让他们办好后，送了过来。

---

string和char数组都是引用类型，引用类型是传地址的，会影响原变量的值，但是string是特殊引用类型，为什么呢？因为string类型的值是不可变的，为了考虑一些内存，安全等综合原因，把它设置成不可变的; 不可变是怎么实现的？Java在内存中专门为string开辟了一个字符串常量池，用来锁定数据不被篡改，所以题目中函数中的str变量和原来的str已经不是一个东西了，它是一个局部引用，指向一个testok的字符串，随着函数结束，它也就什么都没了，但是char数组是会改变原值的

1>创建时的区别：
Statement statement = conn.createStatement(); PreparedStatement preStatement = conn.prepareStatement(sql); 执行的时候: ResultSet rSet = statement.executeQuery(sql); ResultSet pSet = preStatement.executeQuery(); 由上可以看出，PreparedStatement有预编译的过程，已经绑定sql，之后无论执行多少遍，都不会再去进行编译，而 statement 不同，如果执行多变，则相应的就要编译多少遍sql，所以从这点看，preStatement 的效率会比 Statement要高一些。虽然没有更详细的测试 各种数据库， 但是就数据库发展 版本越高，数据库对 preStatement的支持会越来越好，所以总体而言， 验证 preStatement 的效率 比 Statement 的效率高

2>安全性问题 这个就不多说了，preStatement是预编译的，所以可以有效的防止 SQL注入等问题，所以 preStatement 的安全性 比 Statement 高

3>代码的可读性 和 可维护性 这点也不用多说了，你看老代码的时候 会深有体会 preStatement更胜一筹

Java语言使用的是Unicode字符集。而ASCII是国际上使用最广泛的字符编码；BCD是一种数字压缩存储编码方法。

抽象方法可以在接口和抽象类里面声明，抽象类可以没有抽象方法

String (byte[] bytes, String charsetName) 通过使用指定的 charset 解码指定的 byte 数组，构造一个新的

String.getBytes(Charset charset) 使用给定的 charset 将此 String 编码到 byte 序列，并将结果存储到新的 byte 数组。

1，新生代：

（1）所有对象创建在新生代的Eden区，当Eden区满后触发新生代的Minor GC，将Eden区和非空闲Survivor区存活的对象复制到另外一个空闲的Survivor区中。

（2）保证一个Survivor区是空的，新生代Minor GC就是在两个Survivor区之间相互复制存活对象，直到Survivor区满为止。

2，老年代：当Survivor区也满了之后就通过Minor GC将对象复制到老年代。老年代也满了的话，就将触发Full GC，针对整个堆（包括新生代、老年代、持久代）进行垃圾回收。

3，持久代：持久代如果满了，将触发Full GC。

`intValue()` 是把Integer对象类型变成int的基础数据类型；

`parseInt()` 是把String 变成int的基础数据类型；

`Valueof()` 是把String 转化成Integer对象类型;

自旋锁, 阻塞锁, 可重入锁, 读写锁, 互斥锁, 悲观锁, 乐观锁, 公平锁, 非公平锁, 偏向锁, 对象锁, 线程锁, 锁粗化, 轻量级锁, 锁消除, 锁膨胀, 信号量

"|"是按位或：先判断条件1，不管条件1是否可以决定结果（这里决定结果为true)，都会执行条件2

"||"是逻辑或：先判断条件1，如果条件1可以决定结果（这里决定结果为true)，那么就不会执行条件2

接口可以去继承一个已有的接口

接口只能被public和默认修饰符修饰，protected也不行

引用数据类型是引用传递（call by reference），基本数据类型是值传递（call by value）

值传递不可以改变原变量的内容和地址---》原因是java方法的形参传递都是传递原变量的副本，在方法中改变的是副本的值，而不适合原变量的

引用传递不可以改变原变量的地址，但可以改变原变量的内容---》原因是当副本的引用改变时，原变量的引用并没有发生变化，当副本改变内容时，由于副本引用指向的是原变量的地址空间，所以，原变量的内容发生变化。

结论：

值传递不可以改变原变量的内容和地址；

引用传递不可以改变原变量的地址，但可以改变原变量的内容；

关于HashMap的一些说法：

- HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。HashMap的底层结构是一个数组，数组中的每一项是一条链表。
- HashMap的实例有俩个参数影响其性能： “初始容量”和“装填因子”。
- HashMap实现不同步，线程不安全。HashTable线程安全
- HashMap中的key-value都是存储在Entry中的。
- HashMap可以存null键和null值，不保证元素的顺序恒久不变，它的底层使用的是数组和链表，通过hashCode()方法和equals方法保证键的唯一性
- 解决冲突主要有三种方法：定址法，拉链法，再散列法。HashMap是采用拉链法解决哈希冲突的。
    - 链表法是将相同hash值的对象组成一个链表放在hash值对应的槽位
    - 开放定址法解决冲突的做法是：当冲突发生时，使用某种探查(亦称探测)技术在散列表中形成一个探查(测)序列。沿此序列逐个单元地查找，直到找到给定的关键字，或者碰到一个开放的地址(即该

地址单元为空)为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。
    - 拉链法解决冲突的做法是： 将所有关键字为同义词的结点链接在同一个单链表中 。若选定的散列表长度为m，则可将散列表定义为一个由m个头指针组成的指针数 组T[0..m-1]。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。T中各分量的初值均应为空指针。在拉链法中，装填因子α可以大于1，但一般均取α≤1。拉链法适合未规定元素的大小。

Hashtable和HashMap的区别：

- 继承不同。 `public class Hashtable extends Dictionary implements Map` `public class HashMap extends AbstractMap implements Map`
- Hashtable中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。在多线程并发的环境下，可以直接使用Hashtable，但是要使用HashMap的话就要自己增加同步处理了。
- Hashtable 中， key 和 value 都不允许出现 null 值。
- 在 HashMap 中， null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 null 。当 `get()` 方法返回 null 值时，即可以表示HashMap中没有该键，也可以表示该键所对应的值为 null 。因此，在 HashMap 中不能由 `get()` 方法来判断 HashMap 中是否存在某个键， 而应该用 `containsKey()` 方法来判断。

- 两个遍历方式的内部实现上不同。Hashtable、HashMap都使用了Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式 。

- 哈希值的使用不同，HashTable直接使用对象的hashCode。而HashMap重新计算hash值。

- Hashtable和HashMap它们两个内部实现方式的数组的初始大小和扩容的方式。HashTable中hash数组默认大小是11，增加的方式是old*2+1。HashMap中hash数组的默认大小是16，而且一定是2的指数。注： HashSet子类依靠hashCode()和equal()方法来区分重复元素

- HashSet内部使用Map保存数据，即将HashSet的数据作为Map的key值保存，这也是HashSet中元素不能重复的原因。而Map中保存key值的,会去判断当前Map中是否含有该Key对象，内部是先通过key的hashCode,确定有相同的hashCode之后，再通过equals方法判断是否相同。

Java把内存分成两种，一种叫做栈内存，一种叫做堆内存。

- 在函数中定义的一些**基本类型的变量**和**对象的引用变量**都是在函数的**栈内存**中分配。当在一段代码块中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量分配的内存空间，该内存空间可以立刻被另作他用。

- **堆内存**用于存放由**new创建的对象和数组**。在堆中分配的内存，由java虚拟机自动垃圾回收器来管理。在堆中产生了一个数组或者对象后，还可以在栈中定义一个特殊的变量，这个变量的取值等于数组或者对象在堆内存中的首地址，在栈中的这个特殊的变量就变成了数组或者对象的引用变量，以后就可以在程序中使用栈内存中的引用变量来访问堆中的数组或者对象，引用变量相当于为数组或者对象起的一个别名，或者代号。

- 引用变量是普通变量，定义时在栈中分配内存，引用变量在程序运行到作用域外释放。而数组＆对象本

身在堆中分配，即使程序运行到使用new产生数组和对象的语句所在地代码块之外，数组和对象本身占用的堆内存也不会被释放，**数组和对象在没有引用变量指向它的时候（比如先前的引用变量x=null时），才变成垃圾，不能再被使用，但是仍然占着内存，在随后的一个不确定的时间被垃圾回收器释放掉**。这个也是java比较占内存的主要原因。
