JAVA初级知识

1. **Java语言的特性**
2. 跨平台(可移植性):一个应用不需要修改就可以在其他平台上使用.
3. 面向对象的语言:java程序以对象作为基本单元(面向对象的三大特性:)

**继承**:通过子类继承父类达到复用复用父类的目的.(注意父类,子类方法与属性的执行顺序)

附加说明:①调用实例化子类方法,(***多态机制***)都是调用实例化子类重写过父类的方法,除了以下情况:**显示使用super.xxxx**或者**子类中不存在该方法**,或者**子类未重写方法**都是**调用父类的方法, ②子类成员变量与父类成员变量同名时,子类成员变量会覆盖/隐藏父类成员变量**

**封装**:将属性和行为结合在一起,定义为一种类型,通常将信息private

1. **多态**:声明形式一致,实现方式不同.(体现方式:方法重载,方法覆盖,多态参数:形参为父类,实参为任意子类对象. A a = new B();//A为父类,B为子类

注意:指向子类的父类引用,它只能访问父类属性和方法,而对于子类中存在但父类不存在的方法,是无法引用的,重载也不行,如果子类重写了父类的某些方法,调用该方法时,必先调用子类的重写方法.

)

3) 垃圾回收机制:对于不在使用的数据，及时释放其所占的内存。后台线程自动完成

4) 健壮性:在编译和运行时,会对可能出现的问题检查

5) 简单性:取消指针,不过goto语句,不支持多继承

**二、面向对象的五大基本原则(S-O-L-I-D)/面向对象的理解**

1. 单一职责原则（Single-Resposibility Principle）:一个类仅有一个引起它变化的原因,[影响因素过多会产生耦合,维护也会麻烦,牵一发而系全身]
2. 开放封闭原则（Open-Closed principle）: 软件实体/需求应该是可扩展的，而不是在在本身修改的。坚持这个原则就必须尽量考虑接口封装,抽象机制,和多态技术
3. 里氏原则( Liskov-Substituion Principle): 子类能够替换父类,能出现在父类出现的任何地方(父类(标注的模范)一般使用接口/抽象类实现), 例如扩展父类接口的新子类的时候可以不影响当前使用,[理解用例:CS游戏,父类/枪支:子类1手枪,子类2步枪….,]
4. 接口隔离原则（Interface-Segregation Principle）: 使用多个专门的接口比使用单个接口要好的多,类不因该依赖不不需要的接口,[理解用例:有一个接口,有吃,工作俩个方法,人的类实现没没问题,但是换成机器人就有问题,机器人呢不用吃,学习也是一样,成功的方法只能用来参考具体的操作还是要结合本身,因材施教一样]
5. 依赖倒置原则（Dependecy-Inversion Principle）: **传**统的结构化编程中，最上层的模块通常都要依赖下面的子模块来实现，也称为高层依赖低层！所以DIP原则就是要逆转这种依赖关系，让高层模块不要依赖低层模块，所以称之为依赖倒置原则！

[理解用例:订单管理系统的订单类/模块(高层),需要依赖具体的数据库类/模块(低层)实现数据读写假如,现在用sqlServer类,那么就就需要在订单类定义一个sqlServer类去实现,如果要用Oracle数据库呢?重新改成oracleServer类,又要使用mysql数据库呢?订单类过度依赖于数据库类,就需要将数据库类抽象化,定义一个用来数据访问的抽象类型DataAccess

3) 垃圾回收机制:对于不在使用的数据，及时释放其所占的内存。后台线程自动完成

4) 健壮性:在编译和运行时,会对可能出现的问题检查

5) 简单性:取消指针,不过goto语句,不支持多继承

1. 面向对象与面向过程的区别？

面向过程：分析解决问题的步骤，利用函数一步一步实现，按一定顺序调用。

面向对象：把问题分解成一个个对象，目的是为了描述整个解决问题的**步骤中的行为**

举例说明：

五子棋。面向过程的设计思路是先分析解决问题的步骤：

1. 开始游戏
2. 黑子先走
3. 绘制画画
4. 判断输赢
5. 轮到白子
6. 绘制画画
7. 判断输赢
8. 返回步骤2
9. 输出结果

用函数实现，依次调用就解决了。

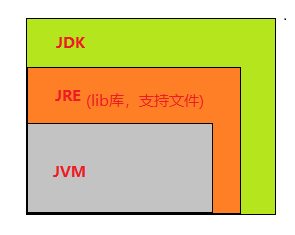
面向对象的实际思路是，整个五子棋可以分为：

1. 黑白双方，行为一样
2. 棋盘系统，负责画画
3. 规则系统，负责判断，输赢，犯规等。
4. JDK,JRE,JVM?

JDK:JavaDevelopKit java开发工具包，包含jre,jvm,能够编译,执行,

JRE:javaRuntimeEnvironment java运行环境,包含jvm,lib,支持文件(对.java文件无法编译)

JVM:java 虚拟机 想象中的一个机器,通过计算机上的软件模拟来实现.

1. Java类的组成元素(5种)?

属性,方法,构造,块,内部类

类初始化过程是这样的：

1.首先，初始化父类中的静态成员变量和静态代码块，按照在程序中出现的顺序初始化；

2.然后，初始化子类中的静态成员变量和静态代码块，按照在程序中出现的顺序初始化；

3.其次，初始化父类的普通成员变量和代码块，在执行父类的构造方法；

4.最后，初始化子类的普通成员变量和代码块，在执行子类的构造方法；

1. 重载,构造方法,块,覆盖 ,抽象方法?

**重载:overload** 同一个类中,方法名相同,参数列表(参数个数,参数类型)不同.

返回值类型不做参考,应为 int fun( return 0;);和 void fun(); 不考虑返回值下直接调用时,不知道调用哪个

**构造:**方法名与类名相同,不声明返回值类型.

**块:**由{ }包起来的代码,实例块(普通代码块):不能直接调用,每次在调用构造之前调用。

**静态块**:只加载一次.类加载时加载,静态代码块>main()>实例块>构造>普通方法

**覆盖**：override/重写 发生在继承关系中，子类使用父类方法,使用@override标注，返回值，方法名，参数列表相同 。方法体可以不同,访问权限不能比父类的更严格，抛出异常范围不能比父类大。

**抽象方法**：没有方法体的方法，使用abstract修饰

1. 访问权修饰符?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | **Public** | **protected** | **default** | **private** |
| 本类 | **√** | **√** | **√** | **√** |
| 同包 | **√** | **√** | **√** |  |
| 子类 | **√** | **√** |  |  |
| 其他包 | **√** |  |  |  |

1. 基本数据类型(8种)?

数值型: {整型:byte/short/int /long 浮点型:float/double} 非数值型: {布尔型:boolean 字符型:char}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据类型** | **字节数** | **范围** | **默认值** | **包装类型** |
| boolean | - | - | false | Boolean |
| byte | 1 | -2^7 ~ 2^7 -1 | 0 b | Byte |
| char | 2 | Unicode 0 ~ 2^16 -1 | ‘\u0000’/null | Character |
| short | 2 | -2^15 ~ 2^15 - 1 | (short) 0 | Short |
| int | 4 | -2^31 ~ 2^31 - 1 | 0 | Integer |
| float | 4 | IEEE754 ~ IEEE754 | 0.0f | Float |
| double | 8 | IEEE754 ~ IEEE754 | 0.0d | Double |
| long | 8 | -2^63 ~ 2^63 - 1 | 0L | Long |

向上转型:由小到大会自动转换, 向下转型:大到小需要强制转换,精度会丢失

Null,可以赋值给引用类型变量,但不能赋值给基本类型变量

1. static,final ,abstract，finally ,finalized?

**satatic**:~~修饰类~~；修饰属性(访问权限 static 数据类型 变量名)，类中所有对象的共享属性,类名.属性/对象名.属性。修饰方法(访问权限 static 返回值类型 方法名)，该方法不属于任何对象，无法直接使用非静态的方法和属性,使用类名/对象名.方法名调用。修饰块(static {..})，类加载时调用，且只调用一次。*修饰内部类(访问权限? static class 类名)*。只能访问外部类的静态成员与方法

**final:**修饰类，该类为一个终极类，无法被继承。修饰属性，属性为常量，赋值后无法改变（声明时赋值/构造时赋值）。修饰方法，方法不能覆盖，可以继承使用。~~修饰块~~，修饰内部类。

**abstract:**修饰类，该类为抽象类：可以有变量，构造，普通方法，抽象方法，无法自己实例化，作为父类，由子类实现。~~修饰属性~~，修饰方法，抽象方法：没有方法体的方法，依靠子类实现。~~修饰块~~，修饰内部类。

1. 抽象类,接口?

抽象类：abstract修饰，无法自己实例化的类，可以有变量，构造，普通方法，抽象方法，被当做父类，子类实现。

接口：interface声明，implements定义，接口中的变量为常量(public static fianl)，方法为抽象方法(public 类型)，无法实例化，不能有static修饰,类与类单继承，一个类可以实现多个接口，接口也可以继承接口 ，如：interface A,B,C; interface A extends B,C。

1. 集合?

**1.Connection<E>接口**：定义了集合类型共同特征，存储的是单个对象。

List接口：按自然顺序排序，元素可以重复，每个元素加入集合的顺序对应一个索引。

实现类：ArrayList：实现动态数组方式存储，元素可以重复 ,查询操作效率高,插入和删除效率低,要考虑数组长度分配,位置移动。LinkedList：实现链表数据结构,插入和删除操作效率高,链表是双向的. Vector：与ArrayList相似，但线程同步。

Set接口：无序，元素不重复，重复元素无法添加。

实现类：

*HashSet*:元素不重复，底层为哈希表结构，根据equals方法（比较对象内存地址）和hashCode方法（比较对象地址的16进制值）比较两个对象是否相同？如果equals方法返回True,则hashCode方法返回值相同。若equals方法返回false,则hashCode方法返回值不一定相同。值可为null

*TreeSet*:保证元素不重复，而且对元素排序,底层是二叉树。TreeSet中的元素必须实现Comparable接口，通过CompareTo()方法(返回一个int值,0相等,参考做减法)比较对象的不同。值不可为null

**2.Map<K,V>接口**：存储的是键值对，key值不重复。

实现类：

***HashMap***:元素不重复(重复的元素(key值)会被后者覆盖)，根据key值的hashCode和equals方法保证元素唯一性，允许空键值对。Key=null,value=null

***Hashtable***:元素不重复(重复元素被后者覆盖)，不允许空key/value值(null)，与Vector相似，线程同步。

***TreeMap***：元素不重复，根据key值排序，key值不能为null,实现Comparable接口，依据CompareTo方法比较对象。

Properties类：HashTable的子类，间接实现了Map接口，主要对属性文件处理。

1. 异常

运行时异常

RuntimeException

检查/**编译**时异常:需要手动try/catch

CheckException

Exception类

Throwable类

Error类

异常：不正常的事件，不是错误。

1. Throw,Throws?

Throw:抛出异常（自定义），在方法体内使用，抛出后必须处理

Throws在方法声明处使用，任何方法都可以使用。

1. Try…catch…finally?

try块:包含可能发生的异常

catch():必须跟在try后，捕获异常后程序结束。

Finally:不过是否会发生都会执行某些代码。先于return语句和catch语句执行

如果**try块有return语句**,**finally块(没有return)会在return语句执行前先执行**,**再执行try块的return语句**

如果try块有return语句,finally块**没有return语句**,finally语句不会修改try块的返回值,依然返回的是try块return语句的值

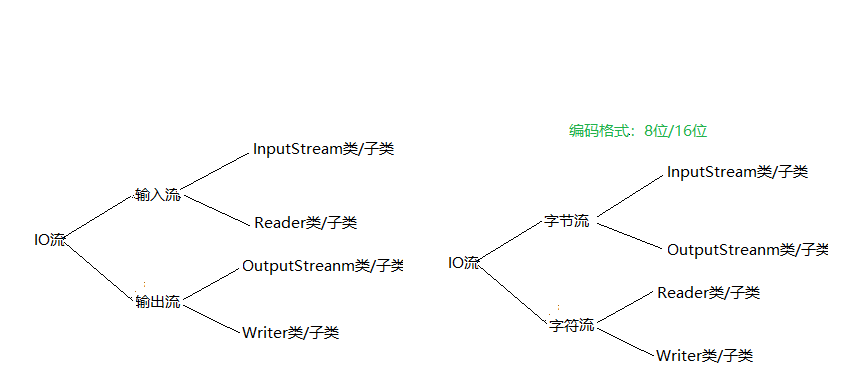
**如果try块有return语句,finally块有return语句**,会直接返回finally块的return语句

Int i =3; try{ i = 4; return i;} finally{ i++; } // i = 4; why?

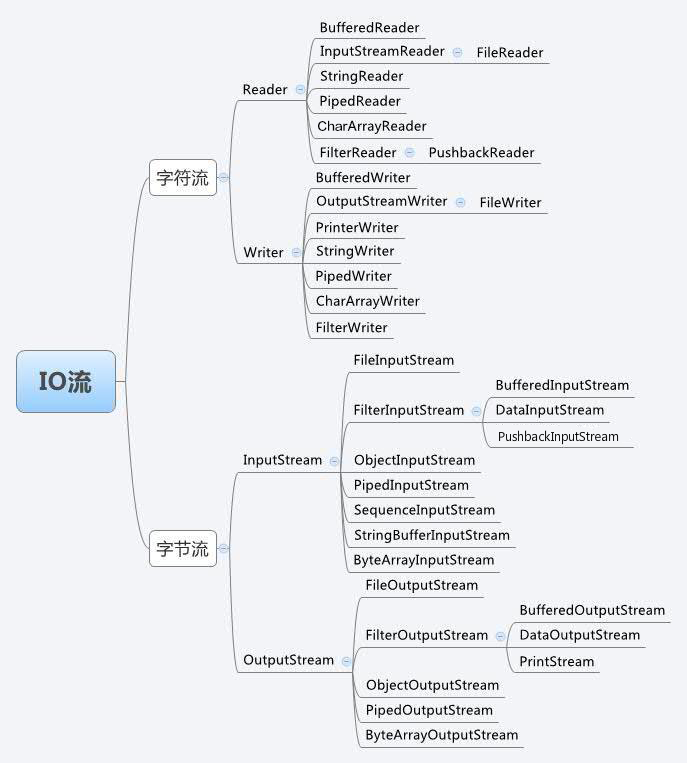
try{}..catch(){} /try{}..catch(){}…catch(){}……finally{}..

1. File类,IO流,序列化与反序列化,枚举?

**File类**：既表示文件也表示目录，



**IO（输入/输出）流**：对数据的读/写操作 。输入input🡪读（read），输出output🡪写(write)



Input类型的Read/读方法🡪程序从数据流(键盘,鼠标,文件)读取信息

举例:

你播放在磁盘D:/妇联4.mv的视频,就是文件的读取

Output类型的Write/写方法-🡪程序将数据写入到数据流中,输出到目的地

举例;

将a.txt文件拷贝到你的硬盘上,就是文件的写入

**字节流**：InputStream/OutputStream 的子类。如：FileInputStream

**字符流：**Reader/Writer的子类。如FileReader

**节点流**：直接封装数据源（File file/File对象）的流类。如FileReader(File file);

**过滤流**：封装流对象(Reader reader)的类。BufferReader(new FileReader());

字节文件流: FileInputStream/FileOutputStream(处理图像等)

字节缓冲输入流: BufferedInputStream/BufferedOutputStream(高效率)

字符转换流: InputStreamReader/OutputStreamReader(字节流转字符流)

字符缓冲流: BufferedReader/ BufferedWriter(高效率)

对象流: ObjectInpuStream/ObjectOutpuStream,处理序列化

使用场景:

字符流: 处理纯文本类型,如txt(**BufferedReader**缓冲字符流),不能处理非文本类型,如音频,图像

字节流: 图像,音频,,ppt,word文档,也可处理文本文件但效率低

**13.序列化**：是将**对象的状态**信息转换为可以存储或传输形式的过程。（持久化的，对象转成数

据格式，字节，xml，，，）ObjectOutpuStream对象流序列化

Transient 英[ˈtrænziənt]:短暂的,临时的,修饰的成员属性,是不会被序列化的, 对象序列化时,由于该属性是短暂的,不会被送到输出流,反序列化时该属性值为0

理解用例:一个三角形类,属性有,长,宽,面积,显然面积能够计算出来,是多余的,没必要序列化,transient修饰,

**反序列化：**将数据格式装换成对象的过程。ObjectInpuStream对象流反序列化

1. nio流与io流的区别?

io流是基于字节流和字符流操作的,一个或多个字节读取,没有缓存的地方.

io流是阻塞模式:使用write()或reade()方法时线程是阻塞的,无法做其他事直到完成为止.

nio流基于缓冲区(Buffer)和通道(Channel)操作的.

nio流是非阻塞模式:一个线程请求写入数据到某通道,不用等到它完全写入,这个线程可以做其他事情.

nio流的选择器可以让一个线程管理多个通道.

1. 实现多线程方法?

继承Thread类 / 实现Runnable接口 /实现Callable接口

1. 进程，线程，线程同步,线程通信方法,死锁?

进程：系统中正在运行的程序

线程：程序中的代码片段，独立执行，完成多个任务。（守护线程：后台执行的线程，垃圾回收的线程。用户线程：程序中创建。使用Thread类中setDaemon(true)可变为守护线程）

区别：不同进程使用不同代码和数据空间，多线程却可以共享数据空间。

**线程同步：synchronized实现，1.同步代码块 2.同步方法 --🡪互斥锁机制:就是只有一个线程可以访问对象**

synchronized修饰**静态方法**以及**同步代码块**的synchronized (类.class)用法锁的是类，线程想要执行对应同步代码，需要获得类锁。  
synchronized修饰**成员方法**，线程获取的是当前调用该方法的对象实例的对象锁

线程通信方法（同步块中使用）：Object中的方法：wait(); notify(); notifiAll();

线程的五种状态:

启动(new),

运行(runnable),

阻塞(blocked),

等待(waiting),

停止(terminated)

1. 泛型,反射,自动装箱与拆箱?

泛型：本质是参数化类型，需要声明数据类型时不具体指定某个类型，而是使用这个参数，具体使用时再指定具体类型。

补充:

泛型类:在类名后使用<字符>声明,字符一般<T>,<G>,<E>…

**public** **class** B <T> {}

泛型方法:在返回类型前声明泛型

**public** **static** <T> String getM(T t) {}

泛型通配符和限定范围:

通配符:?

上限:extends 限定类型,即你的泛型层次不能超过限定类型

下限:super 限定类型,即你的泛至少是限定类型层次

**public** **void** getData(List<? **extends** Number> list) {

//List<?>类型上限是Number类型,不能超过Number范围,如写String

}

父类A,子类B,C 类B的子类D,代表具体对象无法赋值的

List<A> != List<B> != List<C> != List<Object>

范围:?所有范围,?extendsA 最大范围A,?superA最小范围A,List即是具体也是所有范围

List == List<A>

List<D> == List<?extends B> == List<?>

1. 反射：一种强大的工具，灵活创建代码，如，获取对象，调用任意对象的任意方法

反射机制:运行状态中,对于一个类或对象都能调用它的属性和方法.这种动态获取信息及调用方法的功能成为java语言的反射机制.

<1> Object类：getclass();

Class<? extends Number> c lz= Number.getClass();

<2> Class类：forName(“完整类名”),静态方法

Class<Driver> driverClz = Class.forName(“com.jdbc.mysql.Driver”);

<3> 类名.class()，返回值为Class对象

Class cla = User.class(); User user = cla.newInstance();

19. 创建对象的方式:

1. New
2. 构造
3. 反射,如Class cla = User.class() User u = (User) cla.newInstance();
4. 序列化
5. 自动拆箱：将封装类型装换为基本数据类型

自动装箱：将基本数据类型装换为封装类型

如: **int** a = 128;

**int** b =128;

Integer a1 = a;

Integer b1 = b;

// i 值在(-128~127)之间是不会创建Integer对象的,会从缓存里获取

System.***out***.println(a1 == b1);//false

int i ;

Integer m = i; -->自动装箱 ,Integer m = Integer.valueOf(i);

int i = m;-->自动拆箱, int I = m.intValue();

21. String,StringBuffer,StringBuilder

答:String是一个不可改变的类,fianl修饰,无法继承 ,每次对String类型操作都等同于产生一个新的String对象,然后指向该对象.尽量不要大量使用拼接,防止产生太多的临时对象,影响性能,有时jvm会对String优化,String = “av”+”dv”+”ss”;会直接优化成 String s = “avdvss”;不存在拼接

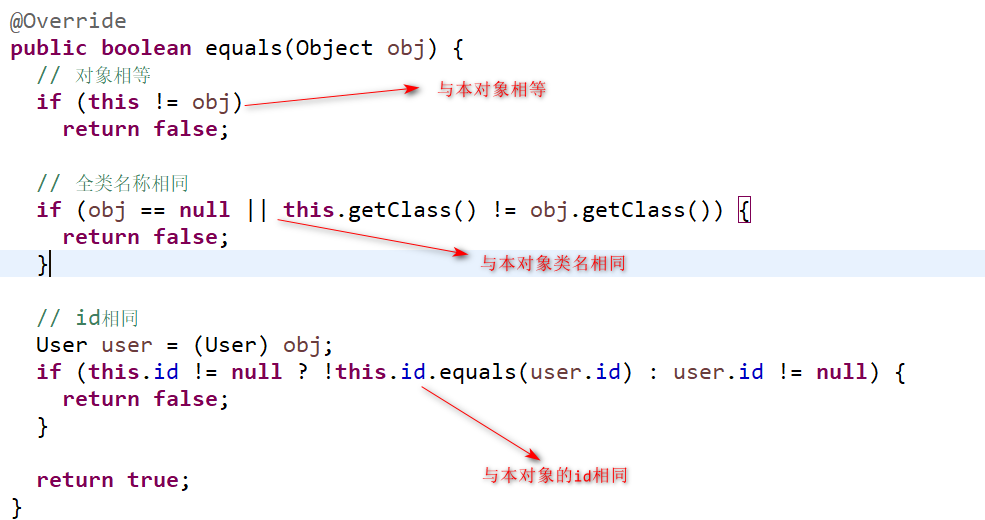
StringBuffer是一个线程安全(使用的Synchronize)可改变的字符串,是对对象本身操作, 常用的方法有追加,插入,删除,替换

StringBuilder是一个线程不安全可改变的字符串,功能与StringBuffer类似jdk1.5加入的,没有使用多线程情况下,建议使用StringBuilder

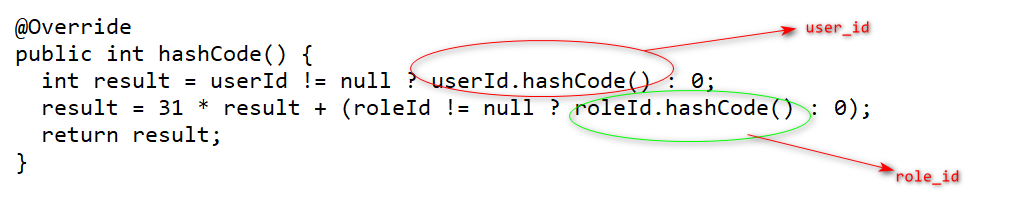
22. sleep( ) 和 wait( ) 有什么区别?

sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。

wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态

23. equals方法的重写, (2个都属于某类的对象,且二者的id相等才是对象相等)

hashCode()方法重写(2个参数的情况下)



1. equals()方法和==的区别?

答:equals方法:--(String类里的)

先比较地址(使用==比较),结果相同, return true

地址不同,再比较类型是为String,不是return false,

是String类型,比较字符串的每一个字符值是否相等

==操作符

如果比较的是基本数据类型,比较的两者的值

如果比较的是非基本数据类型,比较的是引用地址(即堆中内存地址)是否相同

String s1 = “hello”;//放在常量池/方法区

String s2 = “hello”;

String s3 = “he”+ “llo”;//已知的字面量,编译时会直接优化,s3=”hello”

String s4 = “he”+new String(“llo”);//创建了对象,不会优化

String s5 = “he”; final String f5 = “he”;

String s6 = “llo”; final Strign f6 = “llo”;

String str1 = f5 + f6;//常量值拼接固定的

String s7 = s5 + s6;//当作变量赋值给s7,变量是不可预测的

s1==s2==s3 ==str1 != s4 != s7

示例:

**public** **class** SystemUtil{

**public** **static** **boolean** isAdmin(String userId){

**return** userId.toLowerCase()=="admin";

}

**public** **static** **void** main(String[] args){

System.***out***.println(*isAdmin*("Admin"));

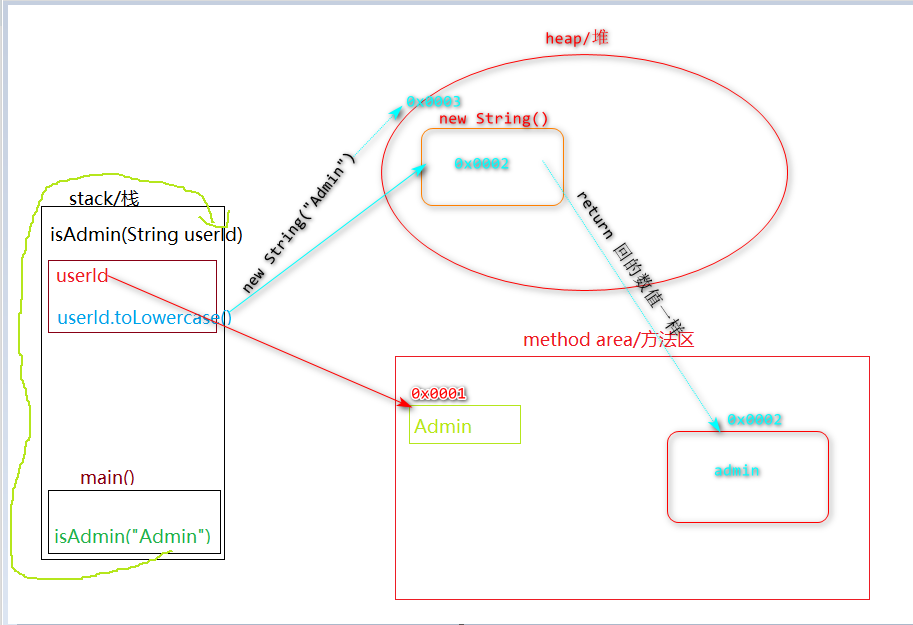
}

}

结果:false

解析:toLowerCase()方法,源码是

return new String(result, 0, len + resultOffset);



25. Match类中3个方法的计算:

1. **Math.ceil**

Ceil中文意思*天花板*,就是表示**向上取接近该值的最大整数**,如Math.ceil(11.3)=12, Math.ceil(-11.6)=-11

2. **Math.floor**

Floor中文意思*地板*,表示**向下取接近该值的最小整数**,如 Math.floor(11.6)=11,

Math.floor(-11.4)=-12

1. **Math.round**

该方法表示”*四舍五入*”,就是Math.round(x + 0.5),即**原来的值先加0.5,再向下取整**,如

Math.round(11.5)=12, Math.round(-11.4)=-11 (-11.4 + 0.5 =-10.9,向下取整 -11)