## java获取class对象的三种方式

//第一种方式： Class.forName()

Class c1 = Class.forName(“Employee”);

//c1代表引用保存内存地址指向堆中的对象，该对象代表的是Employee整个类

//因为在中文API中，forName()方法具有异常，所以主方法要抛出异常

//第二种方式：java中每个类型都有class属性

Class c2 = Employee.class;

//第三种方式： java语言中任何一个java对象都有getClass()方法，getClass()返回运行类

Employee e = new Employee();

Class c3 = e.getClass();

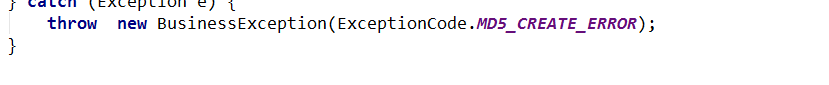
Object的getClass()

静态加载.class

public static void main(String[] args) {  
ClassA a=new ClassA();  
System.out.println(a.getClass());  
System.out.println(ClassA.class);  
}

2.Try catch 报错不会退出程序

## enum类的使用



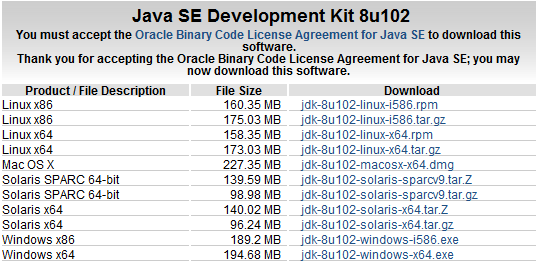
像这样声明一个枚举类型即等同于new一个实例触发构造函数

## Java异常机制

# 入门----Java程序环境配置与启动

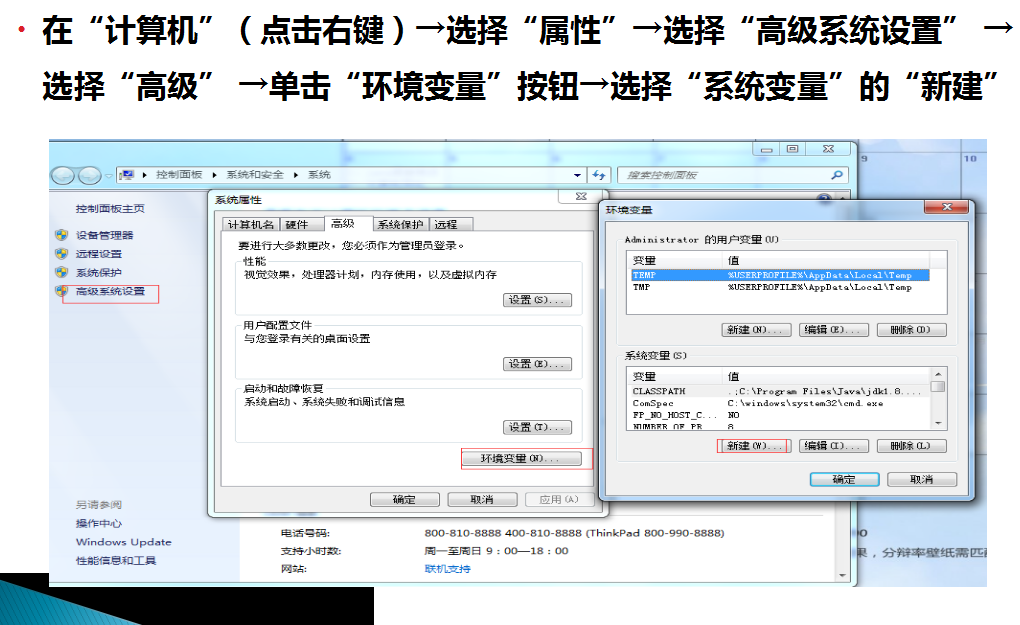
## 1.1下载并安装JDK（http://java.sun.com）。

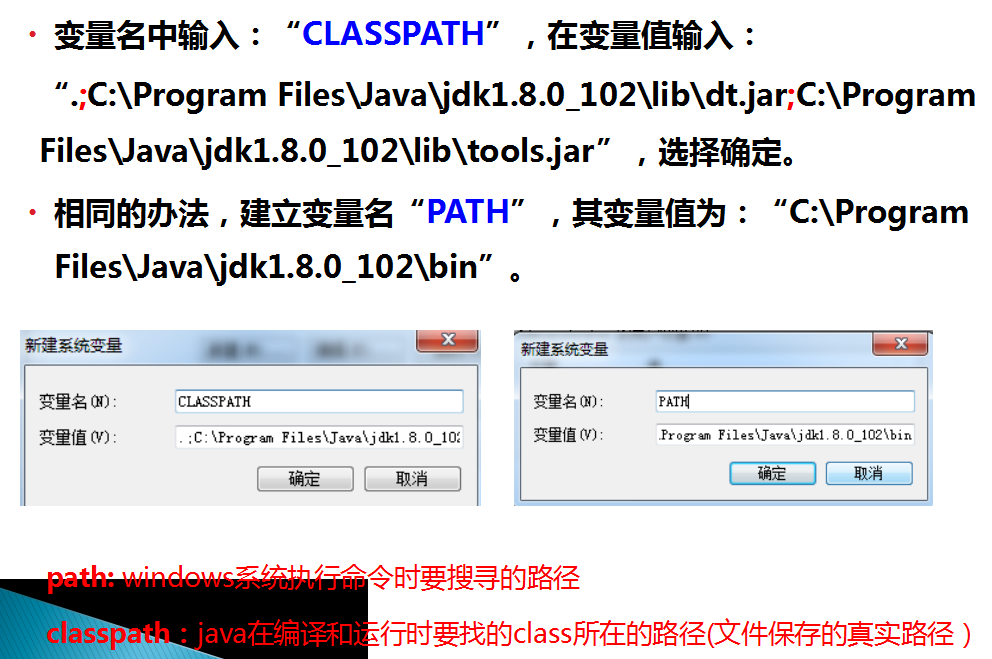
http://www.oracle.com/technetwork/java/javase/downloads/index.html





## 1.2设置环境变量。





## 1.3编写Java Hello World程序。

略

## 1.4在命令行窗口使用命令编译该程序，并查看其运行结果。

命令行查看jdk是否安装成功：java -version

Javac -version

原来文件hello.java

命令行运行Javac hello.java----生成hello.class

命令行运行 java hello 命令行输出

# 二．Java基本语法

标识符

关键字

常量与变量

Java基本数据类型

表达式

**三目运算符**：

Scanner scanner=new Scanner(System.in);  
int a=scanner.nextInt();  
int b=scanner.nextInt();  
int c=scanner.nextInt();

int s=a>b?a>c?a:c:b>c?b:c;  
System.out.println(s);

运算符

程序控制语句

程序格式

## 2.1杂项

### 2.11保留两位小数

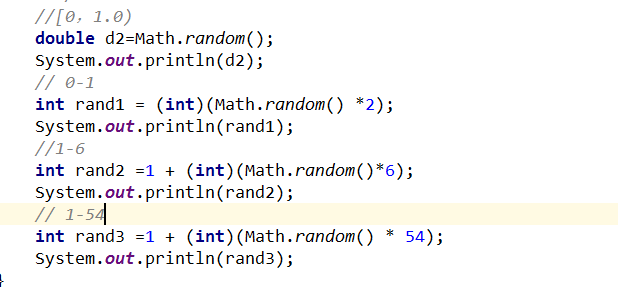
double d = 3.2915926;  
//第1种方式  
 String s = String.format("%.2f",d);  
 //第2种方式   
 DecimalFormat df = new java.text.DecimalFormat("#0.00");  
 String s= df.format(d);  
// 第3种 方式  
 NumberFormat ddf1=NumberFormat.getNumberInstance() ; ddf1.setMaximumFractionDigits(2);  
 String s= ddf1.format(d);  
// 第4种 方式   
 BigDecimal b = new BigDecimal(d);  
 double s = b.setScale(2,BigDecimal.ROUND\_HALF\_UP).doubleValue();  
 System.out.println(s);

### 2.12键盘输入

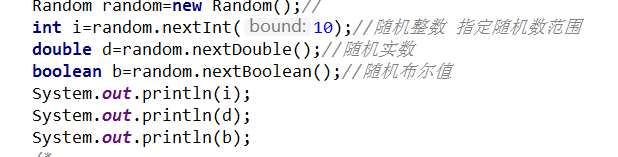
Scanner scanner=**new** Scanner(System.***in***);  
**int** i=scanner.nextInt();*//输入整数***double** d=scanner.nextDouble();*//输入浮点类型*String s=scanner.next();*//输入字符串*System.***out***.println(i);  
System.***out***.println(d);  
System.***out***.println(s);

### 2.13随机数

\* 使用Math类的random方法  
\* Math.random()默认产生大于等于0.0且小于1.0之间的随机double型随机数  
\* 强制转换成int后整数部分赋值给int类型变量，小数点之后的小数部分将会丢失。  
\*如果要生成[0,n]的随机整数的话，只需要Math.random()乘以n+1，生成[0,n+1)的浮点数，  
\* 再强制类型转换为int类型，只取其整数部分，即可得到[0,n]的整数。



*使用java.util.Random*

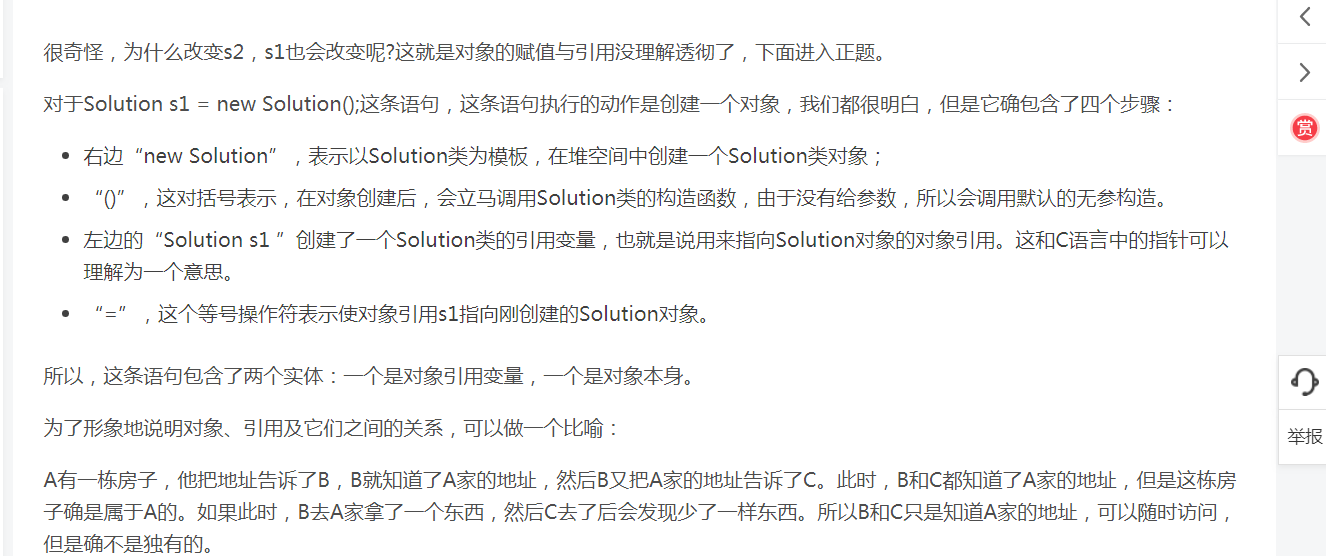


### 2.14java对象赋值是引用

A a=new A();

A b=new A();

改变B中的S1,a中的S1也会跟着改变



## 2.2编程

## 2.3链表

Node类

Link类

#### 定义node节点类

**package** NodeList;  
  
**public class** Node {  
 **public** NodeData **nodeData**;*//保存数据* **public** Node **next**;*//用于保存下一个节点* **public** Node() {  
 **this**.**nodeData**=**null**;  
 **this**.**next** = **null**;  
 }  
  
 **public** Node(NodeData nodeData) {  
 **this**.**nodeData** = nodeData;  
 **this**.**next** = **null**;  
 }  
  
 **public** NodeData getNodeData() {  
 **return nodeData**;  
 }  
  
 **public void** setNodeData(NodeData nodeData) {  
 **this**.**nodeData** = nodeData;  
 }  
  
 **public** Node getNext() {  
 **return next**;  
 }  
  
 **public void** setNext(Node next) {  
 **this**.**next** = next;  
 }  
}

#### 定义Link链表类

**package** NodeList;  
  
**public class** LinkList {  
 Node **head**=**new** Node();*//定义头结点  
  
  
 /\*  
 \* 单链表常见操作  
 \* \*/  
 //头插法添加链表  
 /\*  
 \* 每新插入一个结点放在头结点的后面  
 \* 创建的链表为逆序  
 \* \*/* **public** Node headAddNodeList(String name,**int** number){  
 NodeData nodeData=**new** NodeData(name,number);  
 Node node=**new** Node(nodeData);  
 node.**next**=**head**.**next**;  
 **head**.**next**=node;  
 **return head**;  
 }  
 *//尾插法添加链表  
 /\*  
 \* 每新插入一个结点都放在尾节点的后面  
 \* 创建的链表为正序  
 \* \*/* **public** Node tailAddNodeList(String name,**int** number ){  
 NodeData nodeData=**new** NodeData(name,number);  
 Node node=**new** Node(nodeData);  
 Node temp=**head**;  
 **while** (temp.**next**!=**null**){  
 temp=temp.**next**;  
 }  
 temp.**next**=node;  
 *//node.next=null;* **return head**;  
 }  
  
 */\*  
 \* 链表遍历  
 \* \*/* **public void** selNodeList(Node head){  
 Node temp=head;  
 **int** cnt=1;  
 **while** (temp.**next**!=**null**){  
 temp=temp.**next**;  
 NodeData nodeData=temp.getNodeData();  
 System.***out***.println(**"第"**+cnt+**"个结点\_名字："**+nodeData.getName()+**"\_\_数字:"**+nodeData.getNumber());  
 cnt++;  
 }  
  
 }  
  
 */\*  
 \* 将数据插入到指定位置  
 \*  
 \* \*/* **public void** insertNodeListByindex(String name,**int** number,**int** index){  
 Node temp=**head**;  
 **int** cnt=1;  
 **while**(index!=cnt&&temp.**next**!=**null**){  
 temp=temp.**next**;  
 cnt++;  
 }  
 NodeData nodeData=**new** NodeData(name,number);  
 Node node=**new** Node(nodeData);  
 node.**next**=temp.**next**;  
 temp.**next**=node;  
 }  
 */\*  
 \* 删除某个位置的链表  
 \* \*/* **public void** deleteNodeListByindex(**int** index){  
  
 **int** cnt=1;  
 Node pre=**head**;*//记录删除的前一个结点* Node cur=pre.**next**;*//记录删除的当前节点* **while**(index!=cnt&&cur.**next**!=**null**){*//找到要删除的结点* pre=cur;*//记录要删除结点的前一个结点* cur=pre.**next**;  
 cnt++;  
 }  
 pre.**next**=cur.**next**;  
 }  
  
 */\*  
 \* 返回链表长度  
 \* \*/* **public void** getNodeListLen(){  
 **int** cnt=0;  
 Node temp=**head**;  
 **while**(temp.**next**!=**null**){  
 temp=temp.**next**;  
 cnt++;  
 }  
 System.***out***.println(**"链表长度为："**+cnt);  
 }  
 */\*  
 \* 头插法链表反转  
 \* 就地反转 别不1->2->3 将1，2指针由2指向1  
 \* head 1 2 3  
 \* head 2 1 3  
 \* head 3 2 1  
 \* \*/* **public void** reverseNodeList(){  
 Node cur=**head**.**next**;  
 Node n=cur.**next**;  
 *//头插法反转结点* Node Phead=**new** Node();*//反转头结点* **while**(cur.**next**!=**null**){  
 cur.**next**=n.**next**;  
 n.**next**=**head**.**next**;  
 **head**.**next**=n;  
 n=cur.**next**;  
 }  
 }  
  
  
 */\*  
 \* 递归反转  
 \* \*/* **public** Node reverseList(Node cur) {  
 **if** (cur==**null**)  
 **return** cur;  
 Node next = cur.**next**;  
 Node new\_head = reverseList(cur.**next**);  
 **if**(new\_head==**null**){  
 new\_head=**new** Node();  
 new\_head.**next**=cur;  
 }**else if**(cur==**head**) {  
 **return** new\_head;  
 }**else** {  
 next.**next** = cur;  
 cur.**next** = **null**;  
 }  
 **return** new\_head;  
  
 }  
}

#### 单链表常用方法

**迭代和递归**

##### 3.1单链表反转

###### A:头插法链表反转==迭代

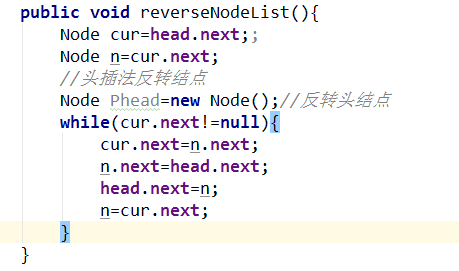
思路：把链表分为两个部分一个为反转部分一个为未反转部分

从前往后开始处理

流程：

*\* head 1 2 3（原链表）  
\* head 2 1 3  
\* head 3 2 1*

具体方法：



java不支持函数参数默认值

###### B：递归

**public** Node reverseList(Node cur) {  
 **if** (cur==**null**)  
 **return** cur;  
 Node next = cur.**next**;  
 Node new\_head = reverseList(cur.**next**);  
 **if**(new\_head==**null**){  
 new\_head=**new** Node();  
 new\_head.**next**=cur;  
 }**else if**(cur==**head**) {  
 **return** new\_head;  
 }**else** {  
 next.**next** = cur;  
 cur.**next** = **null**;  
 }  
 **return** new\_head;  
  
}