命名规则：

规则:字母、数字、符号\_$

规约：类：每个单词首字母大写

属性/方法/变量：驼峰式

test testOne testOneTwo

构造方法：与类一致

静态常量：全部大写，\_做说明

BOOKSTORE\_ADMIN

包名：全部小写，java关键字都是小写，注意不要冲突

1. 数据类型
2. 基本数据类型：8

整型4：byte(字节) short（短整型） int long

浮点型2：float (单精度) double(双精度)

字符型1：char

布尔型1：boolean

1. 引用数据类型：

数组[]

类class（抽象类abstrct class）接口interface

1. 细讲

整型

1. Byte:字节型，每个字节由8个bit(敏感单元位)表示

1byte=1字节= 8bit, 00000000 256种组合，0/1占每个单元位

用第一个bit用来记录正负数，0正，1负数

数值的范围 -2的7次方~2的7次方-1 🡪-128~127

1. short = 2字节=16bit

数值的范围 -2的15次方~2的15次方-1

1. int = 4字节=32bit

数值范围 -2的31次方~ 2的31次方-1

1. long = 8字节= 64bit

数值范围 -2的63次方~ 2的63次方-1

1. float = 32bit由三部分：正负号、整数部分、小数部分
2. double=64bit

注意：

1. 如果存储float e =3.4; 将会报错。

由于计算机🡪内存🡪方法区（类模板）🡪常量缓冲区存储的整数都是32bit（int），而小数都是64bit（float）;

float e =3.4 —>是将3.4（64bit）赋值给e,而e是float(32bit)，会有数据丢失。所以在给float赋值小数的时候规定后面加f/F；

即float e =3.4F;

1. 但是为什么 byte a=1；可以呢？

由于整数是0 0000000000…；

而小数是0 00000000 0000…；

1是32位，前面的都是000，=号在做运算的时候会自动删除前面多余的0，只剩下最后00000001；即8bit；

Byte a = 1;

1🡪32bit;

a🡪8bit;

c．long类型如果真的超过32bit🡪-2147483648~2147483647，需要在后面加l/L；

由于计算机默认数字的整数最多使用32bit，如果超过32bit就会报错，人为加l/L；

1. char

用单引号表示的都是char型，及1char=1字节=8bit，但中文是双倍，16bit

即1char=16bit=2byte 2的15次方-1 0~65535-1

1. boolean

true false 占1个bit

1. Java常量与变量
2. 常量：程序在运行过程中固定不变的值，不能发生改变

作用：固定的值 计算过程中经常用到

1. 命名规则
2. 变量：字母（大小写敏感）、数字（不允许开头）、符号（英文符号\_$）
3. Class:首字母小写、之后的单词首字母大写
4. 计算机工作原理

public class star{  
 public static void main(String[] args){  
 byte x;  
 x = 1;  
 }  
}

1. 计算机包括：内存、硬盘、CPU…

写了以上一段代码，a.在硬盘创建star.java的文件； b.文件中编写以上的代码；c.JVM（javac）将以上的文件编译为star.class; d. 运行（在内存）e.在内存中执行指令

1. JVM运行步骤：
2. 在内存里开辟出一块地方用来执行star.class
3. 这块内存分为三部分：栈内存(弹夹)、堆内存、方法区（类模板）

方法区分为：常量缓冲区、静态元素区static、方法区(类模板)

1. 执行过程：类加载器classloader将硬盘中的字节码（star.class）搬运到内存中。

文件有main、byte x、1；

首先classloader将信息搬运到方法区(类模板)

main要运行，在栈内存里，再创建byte x；而1存储在常量缓冲区；将常量缓冲区的1复制到栈中给x；

1. 数据转换
2. 基本类型

//number - number

byte x = 1;  
int a = x;//correct  
  
int b = 1;  
byte y = (byte) b;//force transition  
// float - double  
float c = 3.4F;//3.4 is 64 bit;  
double z = c;  
  
double d = 4.5;  
float n = (float) d;//force transition

大转小需要强制抓换。小转大不需要

int b = 1000;  
byte y = (byte) b;//force transition  
System.out.println(y);

-24

结果是-24，因为byte类型是2的7次方-1，所以最大127.因此1000超过了8bit；按照最后8个bit存储

//float - int

float a = 3.9F;  
int b = (int) a;  
System.out.println(b);

3

小数转换直接切掉小数点后面的

char x = 'a';  
int y = x;  
System.out.println(y);  
97

字符转换int，在ASCII码，a是97；

int m = 97;  
char n = (char) m;  
System.out.println(n);  
a

int转换字符

总结：大数据可以直接接受小数据，不需要强制转换

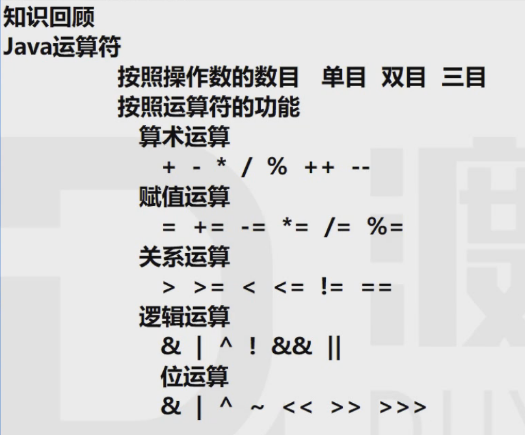
小数据接受大数据需要强制转换，但要注意范围，以免丢失数据

byte a = 3;  
double x = a;  
System.out.println(x);  
  
float m = 3.33F;  
byte n = (byte) m;  
System.out.println(n);

1. Boolean 类型只有一个bit，无法转换；

在Java中，boolean与数字没有联系

1. 运算符



赋值符号：+= -= \*= /= %=

算术运算：+ - \* / % ++ --

关系运算：< <= > >= != ==(判断前后值) instanceof

逻辑运算：& | ！ ^(异或) &&(短路与) ||(短路或)

(3>2) ^ (3>4) 🡪true 只要前后不一致则为true

(3>4)&&(3>2) 🡪 false 如果&&前面的结果为false，则后面的计算短路，不进行计算。

(3>1) || (3>4) 🡪true 如果||前面的结果true，后面的不计算，结果已经确定。如果前面的为false，则计算后面的，前面的忽略不计。

进制转换：以十进制60为例

二进制：8个bit为一个单位，只有0和1

00000000 … 00111100

十进制：60，32个bit为一个单位

=1\*25+1\*24+1\*23+1\*22+0\*21+0\*20

八进制：三个bit为一个单位，以0开头

氛围：01234567

…000 000 000 000 111 100

1. 7 4

十六进制：4个bit一个单位0X3C

范围：0123456789ABCDEF

10A 11B 12C 13D 14E 15F

0000 0000 … 0000 0011 1100

0 0 0X 3 C

进制抓换运算：

System.out.println(3&5);

1

3，5是10进制，分别转换成2进制：

3: 0 0 0 0 0 0 1 1

5: 0 0 0 0 0 1 0 1

& 0 0 0 0 0 0 0 1

结果为1\*20=1

System.out.println(3|5);

7

3: 0 0 0 0 0 0 1 1

5: 0 0 0 0 0 1 0 1

| 0 0 0 0 0 1 1 1

结果为1\*22+1\*21+1\*20=7

System.out.println(3^5);

6

3: 0 0 0 0 0 0 1 1

5: 0 0 0 0 0 1 0 1

^ 0 0 0 0 0 1 1 0

结果为1\*22+1\*21+0\*20=6

取反：

源码：-6—>10000000 …00000110

反码：11111111… 11111001

补吗: 10000000…11111002->11111010

正数的源码、反码、补码都一致

计算机存储过程都是存储补码

System.out.println(~7);//result is -8  
//positive number's source code,inverse code,cpmplement code  
//source code: 00000000 00000111  
//positive number get the negative code,which should get a initial code  
//initial cod: 11111111 11111000  
//initial cod - 1 = qufan  
// : 11111111 11110111  
//get negative again  
// :10000000 00001000  
// 10 bit = -8

System.out.println(~-6);//result is 5  
//source code: 10000000 00000110  
//inverse code:11111111 11111001  
//cpmplement is inverse code + 1  
//cpmplement :11111111 11111010  
//qufan = the qufan of cpmplement  
//qufa :00000000 000000101 = 5  
//负数取发~ = 补码的反码，并且这次反码的时候要改变标志位  
//正数取反~ = 正数的临时反码 - 1，，再最后一次反码

位移：>> >>> << <<<

左位移:6<<1🡪00000110 变成 00001100

右位移:6>>1🡪保持前面的符号位，其他的右移，前面填0

>>>🡪不保留符号位

2<<3相当于2的3次幂

1. ++

int x = 2;  
int y = x++;  
System.out.println("y = : " + y);  
System.out.println("x = : " + x);

2

3

先赋值，再自增

int m = 2;  
int n = ++m;  
System.out.println("n = : " + n);  
System.out.println("m = : " + m);

3

3

先自增，再赋值

1. 其他+=

byte x = 1;  
x = x + 1;  
System.out.println(x);

报错：从int转换到byte可能会有损失

x🡪00000001

常量1是32bit🡪00000000 … 00000001

+号会自动给x填充24个0；

运算完结果是0000000 … 00000002；

但=号赋值的时候等号前面是bit，但后面是表达式，由于不知道后面的编译运算结果bit是否可以自动省略，所以报错；

如果=后面是常量，可以直接添加。

int x = 1;  
x = (byte)x + 1;  
System.out.println(x);

强制抓换不会报错，但有丢失数据风险

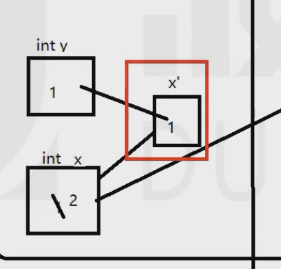
byte x = 1;  
x += 2;  
System.out.println(x);

+=是一个符号，直接加减后赋值

1. 数据在内存中的计算过程

int x = 1;  
int y = x++;

result y = 1;



计算机已经在内存中创建了int x；在x参与计算的时候会把x进行备份x’，计算机计算顺序，计算符号优先级高于赋值符号。X++为2，及x = 2；但赋值的时候会直接赋值给y的值是x’；因此y = 1；

(2). x = x++

int x = 1;  
x = x++;

result = 1

内存中先确定了x=1，运算优先于赋值，先备份x’为1，x++为2，x的值此时为2，x给x赋值，使用的是x’的值，所以此时x的值还是1；

(3). Y = ++x

int x = 1;  
int y = ++x;

result = 2

1. 语法结构
2. 分支：

(1).if

(2). switch case

Switch(x){

Case 1:

System…..

Case 2:

System….

Default:

System….

}

1. **循环**

**For:**

**break🡪结束for循环**

**continue🡪结束此次循环，进行下一轮for循环**

1. **for(初始值;循环结束的条件;循环操作){**

**}**

1. **初始值**

**for(;结束条件;循环操作){**

**}**

1. **初始值**

**for(;结束条件;){**

**循环操作**

**}**

1. **标识符：for(;;){**

**}**

**While**

1. **初始值**

**While(循环截止的条件){**

**循环操作**

**}**

**do{…}while();**

1. **初始值**

**do{**

**循环操作;**

**}while(循环截止);**

****

1. **数组**
2. **定义：数组是一个数据类型相同的集合，将这些数据统一管理起来**
3. **静态初始化：说明长度、有元素**

**int Array[] = new int[]{10,0,5,4,8,44};**

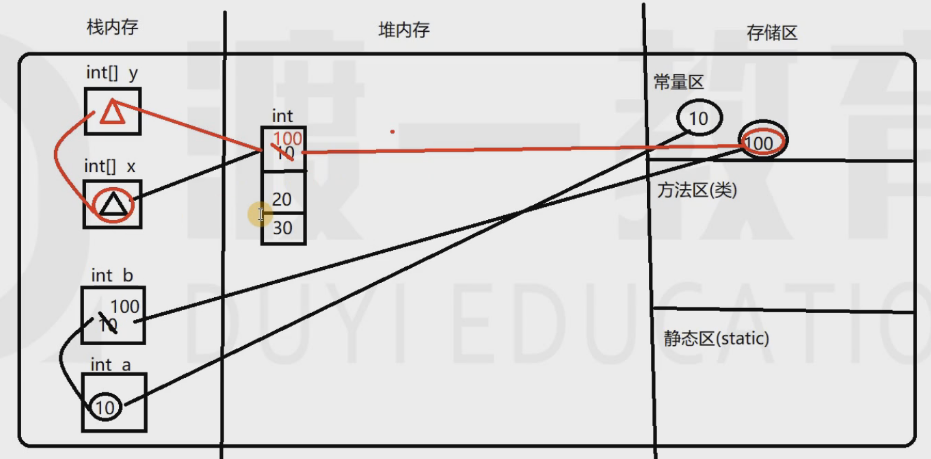
**int Array[] = {2,3,4,9,5,555,1};**

**动态初始化：只有长度,没有元素（其实都是默认元素）**

**int Array = new int[5];**

**默认：int🡪0 浮点—>0.0 char🡪’0’对应的uncode码**

**String🡪null boolean🡪false**

****

**数组存在于堆内存中，数组的名称存在于栈，分配了识别码**

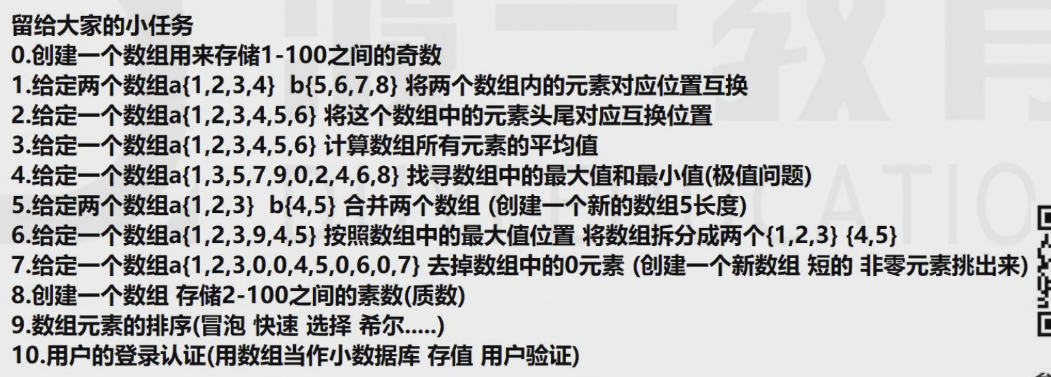
int[] x = new int[]{10,20,30};  
int[] y = x;  
y[0] = 1000;  
System.out.println(x[0]);

🡪1000

**计算素数**

for (int i = 2; i < 100; i++) {  
 boolean x = false;  
 for (int j = 2; j <= i - 1; j++) {  
 if (i % j == 0) {  
 x = true;  
 break;  
 }  
 }  
 if (x == false) {  
 //System.out.println("su shu: " + i);  
 }  
}

**数组任务及作业代码：**

****

**作业：**

****

1. **数组其他用法：**

**Arrays.sort(a);//排序，从小到大**

1. **二维数组**

**静态初始化：**

**int[][] Array = {{1,5,32,9},{0,1},{33,58,7}};**

**Array[0] = {1,5,32,9};**

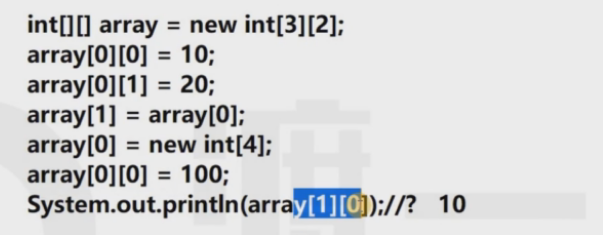
**Array[1][0] = 0;**

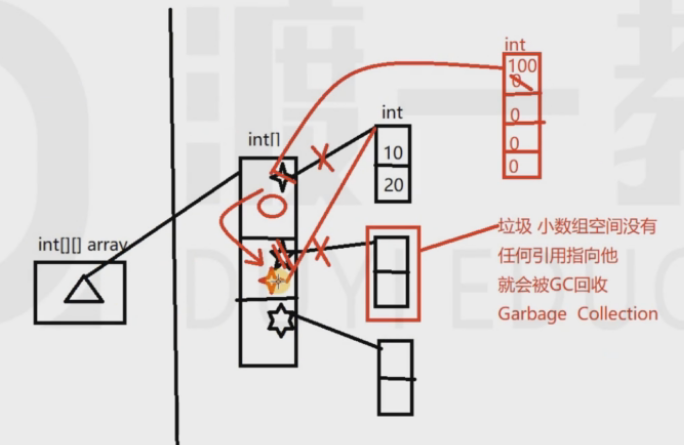
int[][] DA = {{1,5,32,9},{0,1},{33,58,7}};  
for (int i = 0; i < DA.length; i++){  
 for (int j = 0; j < DA[i].length; j++) {  
 int v = DA[i][j];  
 System.out.print(v + "\t");  
 }  
 System.out.println();  
}

**动态初始化：**

int[][] DA = new int[3][2];  
//DA --> {{x,x},{x,x},{x,x}};

**问题：**

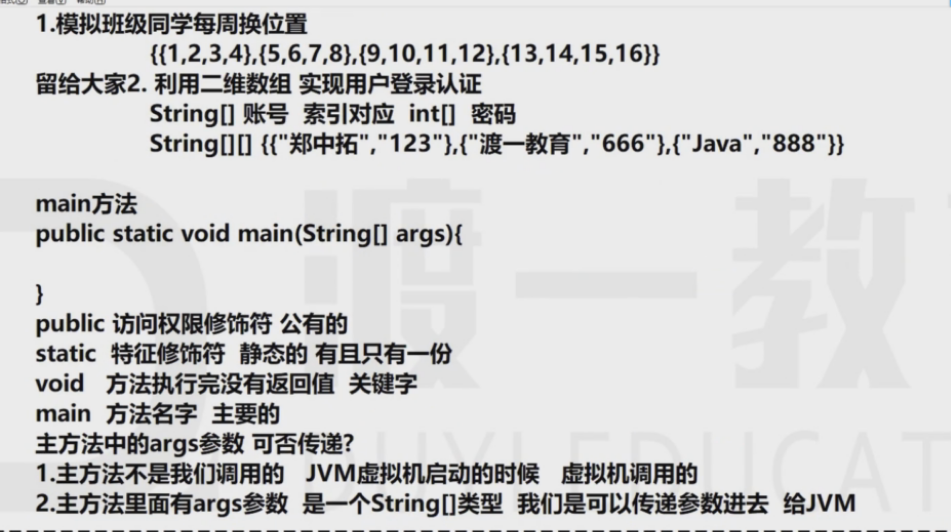




解释：1. Int[0] = {10,20}; 给int[1]也分配{10,20}

2.新的数组int[0] = {100,0,0,0},但int[1]还是之前的数组

3.因此int[1][0]=10;



1. OO

类：描述某一类事物，有具体的共有特征—>人类

静态特征：名字、年龄🡪属性

动态特征：吃饭、学习🡪方法

对象： 类中的具体的某一个个体🡪Moso

public class TestClass {  
 String name;

//修饰符 数据类型 熟悉名字  
 public String Name;  
 public static void main(String args[]){  
   
 }  
}

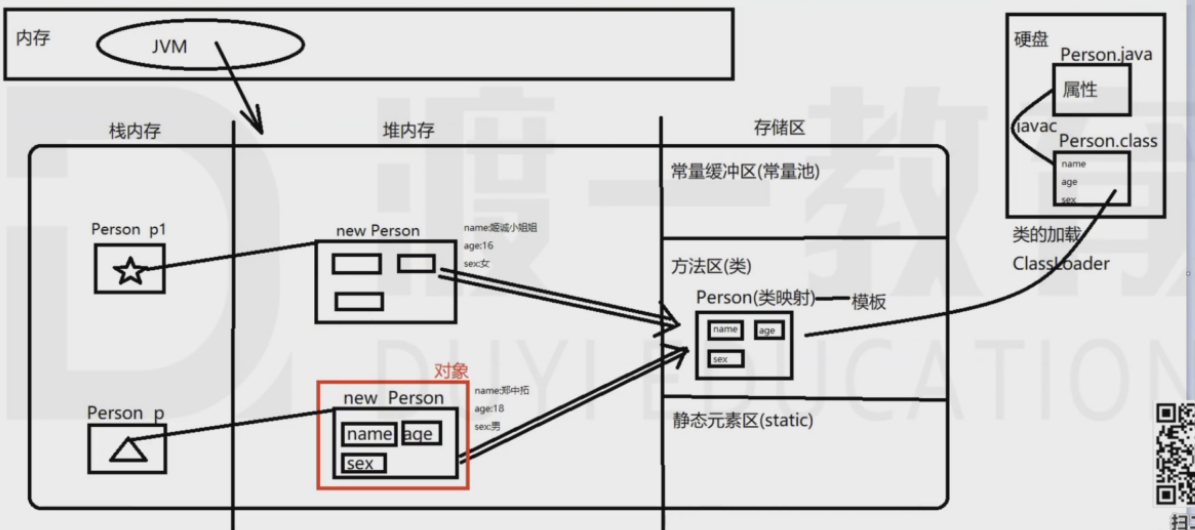
如果不写public，默认修饰符只有在本类和同包下有访问权限；

新建了一个类Person：分配了3个属性

public class Person {  
 public String name;  
 public int age;  
 public String sex;  
}

新建了2个对象p与p1：

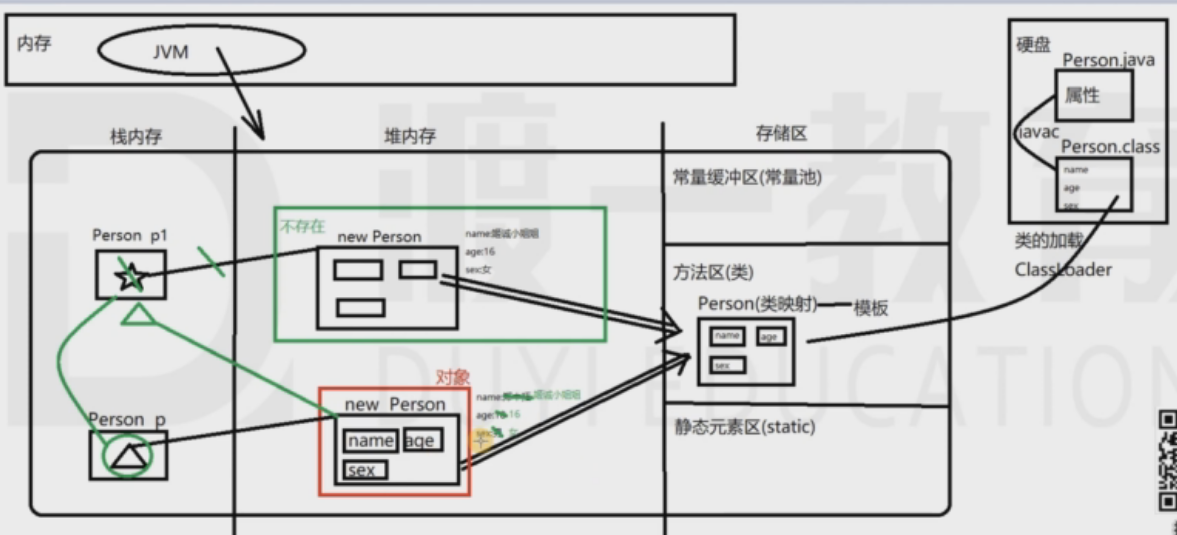
Person p = new Person();  
p.name = "王Moso";  
p.age = 18;  
p.sex = "男";  
Person p1 = new Person();  
p1.name = "Moso王";  
p1.age = 16;  
p1.sex = "女";



如果

Person p1 = p;

则按照下图：



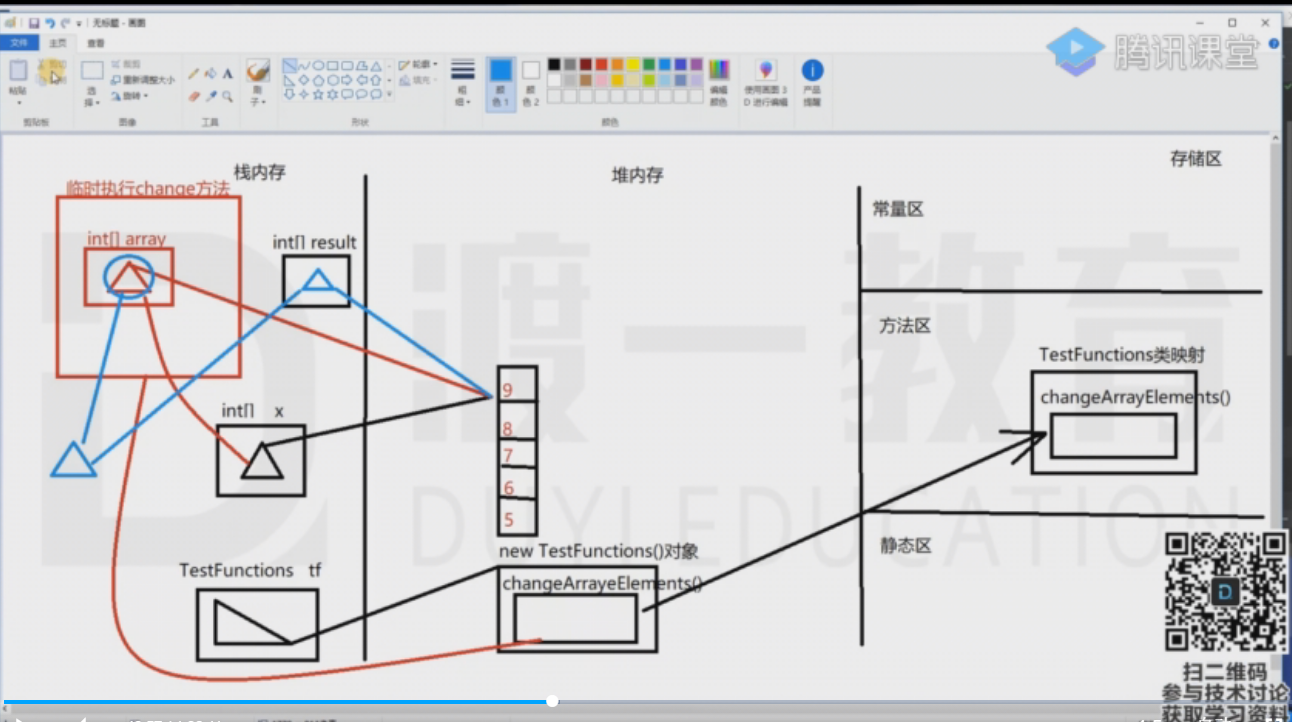
我是XX, 我今年16岁，我的性别是女

我是XX, 我今年16岁，我的性别是女

//每一个类不是必须包含主方法

//主方法不属于任何一个类，主方法属于虚拟机

通过传参的方式翻转Array的过程图：



1. 面向对象之重载

同一个方法名，传参不同。实现重载

1. 面向对象之---构造方法

用于创建当前类的对象

写法：权限修饰符 与类名一致的方法名(参数列表) [异常]{

方法体

返回对象

}

用法：通过new调用

Person p = new Person(); 默认隐形存在无参构造方法

构造方法可以重载

1. This的用法

定义：是一个关键字(代词) 代替的是某一个对象(当前调用属性或方法时的对象)

在同一个类中调用其他的方法或属性

构造方法可以调用构造方法，但一般方法不能调用构造方法，使用this调用构造方法；this();而且要放在第一行

public class Person{   
 public String name;  
 public int age;  
 public Person(String n, int a){  
 this.name = n;  
 this.age = a;  
 System.*out*.println(name + "今年" + age + "岁");  
 }  
}

1. 程序块

写法：{

程序内容

}

用法：块需要调用才能执行，在执行构造方法之前会自动调用程序块；

1. 数据类型转换
2. Int—>Integer char—>Character float 🡪 Float

String🡪int 例： int num = Integer.paraInt(String “123”);

1. Scanner input1 = new Scanner(System.in);

int a = input1.next(); 🡪读取空格或回车前的内容

Int a = input1.nextInt(); 🡪读取回车前的内容，回车符保留

String a = input1.nextLine(); 🡪读取回车前的内容，并且将回车符销毁

1. String不能用==，应该使用a.equil(b)

计算器小练习：



ArrayBox作业

1. **类之间的关系**

**类之间的关系：**

**A is-a B 泛化(继承 实现)**

**A has-a B 包含(组合 聚合 关联)**

**A use-a B 依赖(依赖) (need –a )**

**（一）、继承**

**1. 子类继承父类 extends**

**2. 子类的对象可以调用父类(public protected)的属性、方法 当做自己的使用**

**3. 子类可以添加自己独有的属性和方法**

**4. 子类从父类继承方法，不能满足时，可以重写(覆盖)父类的方法:保留结构、修改内容**

|  |  |  |
| --- | --- | --- |
|  | 方法重写 override | 方法重载 overload |
| 1.类 | 产生两个继承关系的类 子类重写父类的方法 | 一个类的方法 |
| 2.权限 | 子类可以大于等于父类 public protected private | / |
| 3.特征 | final 子类不能重写 static 子类不存在 abstract 子类必重写 | / |
| 4.返回值 | 子类可以小于等于父类 | / |
| 5.名字 | 一致 | 一致 |
| 6.参数 | 一致 | 必须不一致 |
| 7.异常 | 编译时异常： 子类抛出异常个数<=父 子类抛出异常类型<=父 运行时异常： 父类抛出，子类不理会 | / |
| 8.方法体 | 子类与父类不一致 | 随便 |

1. **继承可以传递**

**每一个类都有继承，如果不写，默认继承Object类**

1. **Java中的继承是单继承，一个类只能继承一个**
2. **继承在内存中的继承方法**
3. **关于this和super**

**This和super都是代词**

**This代替的是当前执行方法/属性的对象，不一定是当前类**

**Super代替的是当前执行类的父类中的方法/属性**

**(2).This和Super都可以调用构造方法**

**并且放在另一个构造方法的第一行，两个只能出现其中的一个**

**(二)、 包含🡪has a**

**包含关系(组合 聚合 关联)**

**三种不同类别：从亲密程度讲不一样**

**组合：人和大脑 人和心脏的关系**

**整体和部分的关系 不可分割 同时出现 同时消失**

**聚合：汽车和轮子 电脑和主板**

**整体和部分的关系 创建时有可能分开**

**关联：人有汽车 人有电脑**

**整体和部分的关系 可以分割 后来形成一起**

**从JAVA程序来描述的关系：通过一个类的对象当做另一个类的属性来存储（轮子当做汽车的属性）**

**(三)、依赖use a**

**屠夫 杀 猪 农夫 养 猪**

**一个类屠夫：**

**方法：杀猪**

**需要：猪 刀**

**不是整体和部分的关系 某一件事情产生了关系**

**临时组合在一起 做完关系及解散**

**b.Java程序体现的形式：**

**一个类的方法使用了另一个类的对象**

**第一个可以在方法传递参数**

**第二个可以在方法中自己创建**

****

1. **修饰符**

**权限修饰符 特征修饰符**

1. **权限修饰符**

**Public > protected > 默认不写default > private**

1. **特征修饰符**

**final 最终的 不可更改的**

**static 静态的**

**abstract 抽象的**

**native 本地的**

**\*transient 瞬时的 短暂的------>序列化**

**\*synchronized 同步的 线程问题**

**\*volatile 不稳定的**

**权限修饰符访问权限：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 权限修饰符 | 名称 | 本类 | 同包 | 子类 | 当前项目，没有任何关系 |
| public | 公共的 | √ | √ | √ | √ |
| protected | 受保护的 | √ | √ | √ |  |
| 默认不写 | 默认的 | √ | √ |  |  |
| private | 私有的 | √ |  |  |  |

**Protected不能跨类使用**

1. **Java类和类之间的关系**

**继承 关联 依赖**

1. **Java面向对象的四个特征**

**继承 封装 多态 抽象**

1. **对属性本身的封装(封装的思想):**

**属性私有(封装在类中)**

**提供操作属性相应的方式(公有的方法)**

**将数据或执行过程包装起来，保护数据或者执行过程的安全（私有）**

1. **修饰符**

**权限修饰符**

**public 公共的**

**protected 受保护的**

**默认不写 默认的**

**private 私有的**

**特征修饰符**

**final 最终的 不可更改的**

**static 静态的**

**abstract 抽象的**

**native 本地的**

**\*transient 瞬时的 短暂的------>序列化**

**\*synchronized 同步的 线程问题**

**\*volatile 不稳定的**

**final 最终的 不可更改的**

**特征修饰符可以修是什么？**

1. **修饰变量：静态变量初始后不能更改**

**引用类型可以修改值，但堆内存的地址不允许更改—>对象唯一**

int[] aArray = new int[]{1,5,4,8,44};  
aArray[0] = 21;

**教室和钥匙的关系**

1. **属性、方法、类本身**

/**/属性与变量的区别，属性在类中单独存在，有权限修饰符。变量在方法中。没有权限  
public final String a = null;//属性**

**属性：必须赋值**

**方法：final修饰的方法不能被子类重写**

**类：只能引用，不能继承**

**特种修饰符之static**

**Static 静态的**

1. **能修饰什么？**

**属性：方法：块：修饰类（内部）：**

1. **特点：**

**每个类加载都在方法区，并且每个类都有自己的静态元素区**

1. **静态元素在类加载时就初始化了，创建的非常早，此时没有创建对象；**
2. **存储在静态元素区**
3. **在类及类的对象只有一份，供整个类共享（一份供N多对象使用）**
4. **静态元素不属于其他，只属于类**
5. **由于静态元素加载的时候还没有创建对象，可以通过类名字直接访问**

**解释:创建对象：new Person();**

1. **内存管理：栈内存创建开始用完即回收 堆内存通过GC回收**

**静态元素区，GC没有权限管理(Garbage collection),认为是常驻内存**

public class Test\_Static {  
 public String name;  
 public static int *age*;  
 public static void main(String[] args){  
 Test\_Static ts1 = new Test\_Static();  
 ts1.name = "wang";  
 ts1.*age* = 16;  
 System.*out*.println(ts1.name + "今年" + ts1.*age* +"岁");  
 Test\_Static ts2 = new Test\_Static();  
 ts2.name = "liu";  
 ts2.*age* = 17;  
 System.*out*.println(ts1.name + "今年" + ts1.*age* +"岁");  
 System.*out*.println(ts2.name + "今年" + ts2.*age* +"岁");  
 }  
}

1. **静态元素中不可以出现this或supper**

****

1. **设计模式**

**Finally 修饰的必须要有初始值**

**23种 (内功心法)**

**1.设计模式不是知识点**

**2.设计模式是一种设计经验的总结**

**3.设计模式用来解决某些场景下的某一类问题的---->通用的解决方案**

**4.有了设计模式之后 可以让代码更容易被理解 确保了复用性 可靠性 可扩展性**

**设计模式分为三类**

**1.创建型模式(5种)----->用于解决对象创建的过程**

**单例模式 工厂方法模式 抽象工厂模式 建造者模式 原型模式**

**2.结构型模式(7种)----->把类或对象通过某种形式结合在一起 构成某种复杂或合理的结构**

**适配器模式 装饰者模式 代理模式 外观模式 桥接模式 组合模式 享元模式**

**3.行为型模式(11种)---->用来解决类或对象之间的交互 更合理的优化类或对象之间的关系**

**观察者模式 策略模式 模板模式 责任链模式 解析器模式 迭代子模式**

**命令模式 状态模式 备忘录模式 访问者模式 中介者模式**

**(一)、单例模式🡪Singleton**

**概念：创建一个对象，堆内存中只开辟一个空间，供很多人使用**

**1. 单例模式的实现：**

**a. 私有的构造方法**

**b. 私有的静态的当前类作为属性**

**c. 公有的静态方法返回当前类对象getSingleton();**

**指的是对象的加载**

1. **饿汉式(立即加载) 对象启动时就加载啦**

**不会产生对象没有就拿来使用的问题 空指针异常**

**启动项目加载的对象过多 有些还没有使用 产生服务器承载压力的问题**

1. **懒汉式(延迟加载) 对象什么时候用到了 才会加载**

**可能会由于没有操作好 导致异常(很严谨)**

**启动项目时候只有需要的加载 不需要的还没有创建 不会浪费空间**

1. **生命周期托管(单例对象别人帮我们处理) 对象加载过程交给别人**

****

**--------------------------类 加载机制----------------------------------**

**存在继承关系的类 加载机制 及执行过程**

**加载类的过程---静态元素已经加载**

**new Person(); 🡪加载类，的同时会加载静态元素**

**1.加载父类**

**2.父类会产生自己的静态空间 静态（属性 方法 块）**

**执行静态块**

**3.加载子类**

**4.子类会产生自己的静态空间 静态（属性 方法 块）**

**执行静态块**

**5.开辟对象空间**

**6.加载父类的非静态成员 属性 方法 块 构造方法**

**7. 执行（非静态）块 执行父类构造方法**

**8.加载子类的非静态成员 属性 方法 块 构造方法**

**9. 执行（非静态）块 执行子类构造方法**

**10.将对象空间的地址引用交给 变量来存储**

**=============================================**

**Native 本地的**

**Java看到native就再看不到其他的底层C、C++**

**hashCode();**

**abstract();🡪抽象，不具体、没有执行、只是概念**

1. **可以修饰什么**

**修饰方法：不能有方法体；**

public abstract void eat(int a);

//native修饰的方法虽然也没有方法体 但是不是抽象方法 只是执行的过程是其他语言写的 看不见

**修饰类：抽象类**

1. **修饰后有什么特点**

**抽象类可以有抽象方法，但是非抽象类不可以；**

**方法： 抽象类中必须有抽象方法么? 不是必须含有抽象方法**

**抽象方法必须放在抽象类中么? 目前来看必须放在抽象类中(或接口中) 普通类是不允许含有抽象方法**

1. **抽象类的使用**

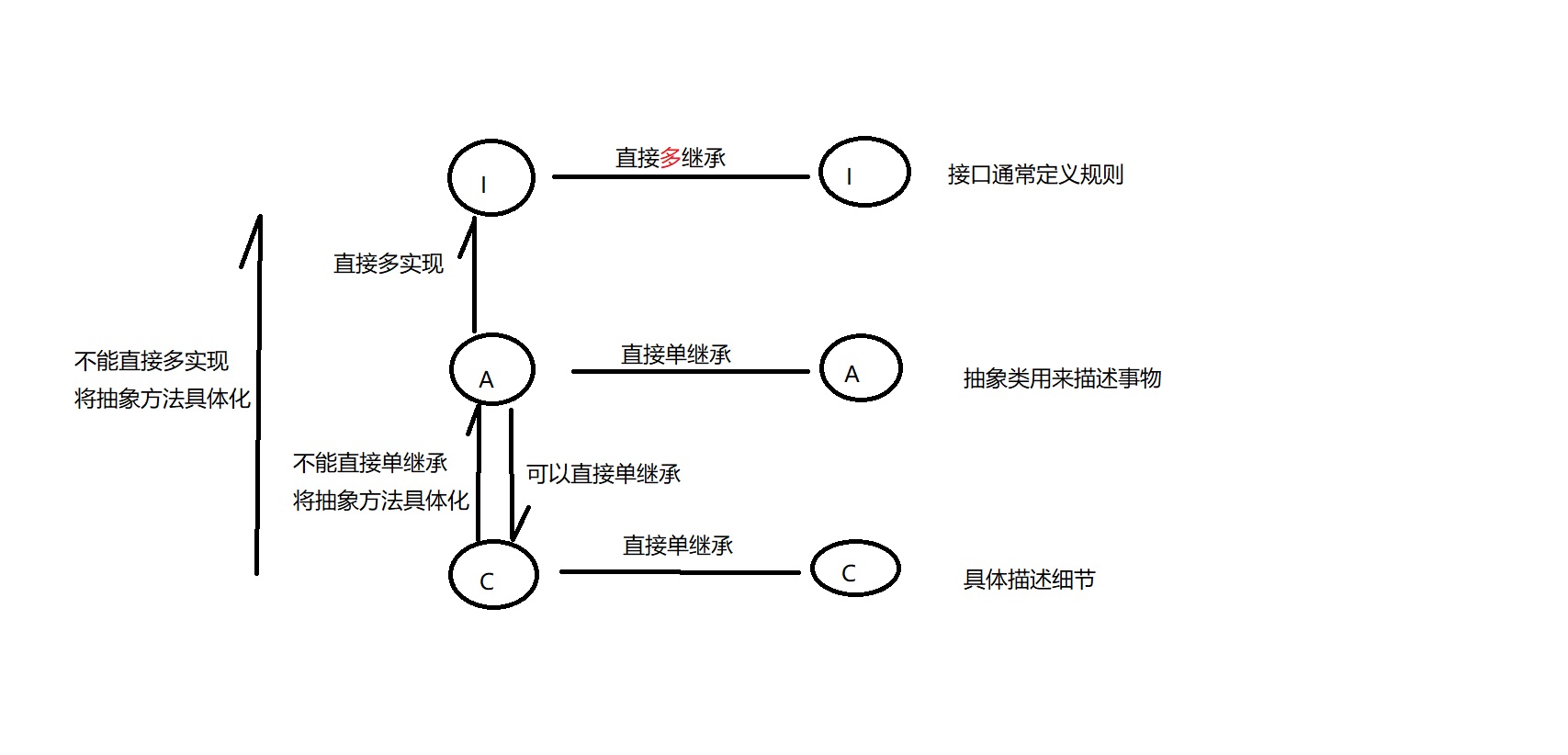
**抽象类只能通过子类单继承来做事**

1. **类和类的关系**

**抽象类----直接单继承----抽象类 可以**

**抽象类----直接单继承----具体类 可以 (用法通常不会出现)**

**具体类----直接单继承----抽象类 不可以 (将父类的抽象方法具体化 或子类Override父类的抽象方法 或子类变抽象类)**



**接口类定义规则；抽象类用来描述事物；具体类用来描述细节**

1. **接口interface**

**抽象类中能不能没有具体成员 全部都是抽象方法 可以 ---> 抽象类抽象到极致 质的变化 ---> 接口**

**接口可以理解为是抽象类抽象到极致--->还是一个类的结构 不能用class修饰 改用interface修饰**

1. **接口有什么**

**属性 不能含有一般属性 只能含有公有的静态的常量 public static final**

**方法 不能含有一般方法 只能含有公有的抽象的方法(1.8 defualt修饰具体方法)**

**不能创建对象**

**只能通过子类多实现(implements)来做事**

**public class A implements B,C,D{**

**}**

1. **抽象类之间的操作**

**抽象类----直接多实现----接口 可以**

**具体类----直接多实现----接口 不可以(必须将接口中的抽象方法具体化 自己变成抽象类)**

**接口是最抽象的，不能继承别的类**

**接口之间可以多继承**

**继承执行顺序🡪执行父类的块🡪父类的构造方法🡪子类的块🡪子类的构造方法**

**类的结构：继承extends、包含、依赖**

**包含（车轮属于汽车，是汽车的一部分）🡪汽车类把轮子类作为自己的属性**

**依赖（警车追击小汽车/记速器测量小汽车通行时间）🡪记速器类计算时间的方法，使用到了小汽车类获取小汽车速度的方法🡪在自己类的方法中，使用另一个类的方法传递参数**

1. **适配器**
2. **缺省适配器**

**将定义的父类中的方法，子类有一部分不需要，加入了缺省适配器的概念**

**父类：**

public interface Box {  
 public boolean add(int element);  
 abstract int remove(int index);  
 abstract int get(int index);  
 abstract int size();  
 abstract String test(String testString);  
}

**适配器：**

public abstract class AbstractBox implements Box {  
 public abstract boolean add(int element);  
 public abstract int remove(int index) ;  
 public abstract int get(int index);  
 //适配器将下面的方法具体化  
 public int size(){  
 return 0;  
 }  
 public String test(String testString){  
 return null;  
 }  
}

**调用：（具体化后的方法不需要强制引用）**

public class TestAb extends AbstractBox {  
 @Override  
 public boolean add(int element) {  
 return false;  
 }  
 @Override  
 public int remove(int index) {  
 return 0;  
 }  
 @Override  
 public int get(int index) {  
 return 0;  
 }  
}

1. **多态**

**Person是People的子类**

**Person p = new People();**

1. **父类类型的引用 ，指向子类的对象**
2. **该引用只能调用父类中有的方法或方法**

**如果是子类新加的，则不能引用**

1. **如果子类中将父类的方法重写，调用后执行的是子类重写的**

**如果子类与父类有相同的属性，调用的是父类的属性**

1. **如果想要调用子类新加的方法或者属性，需要强制转换**

**Teacher t = (Teacher)p;**

**这个强制转换只能针对自己独有的方法或者属性，其他方法还是从底层开始找**

1. **前面的类(Person) >= 后面的类(Teacher)**
2. **如果多级extends，则先在最底层的子类中开始逐级往上找**
3. **ClassCastException 造型异常，将对象的类型还原时，与真实类型不匹配**

**方法从最底层的类开始往上找，参数从最高层往下找。**

**如果想要避免造型的异常 可以用instanceof关键字来进行判断**

**对象 instanceof 类**

**Object o = new Teacher();**

**if(o instanceof Student){//对象是否属于后面类型**

**System.out.println("类型匹配 可以造型");**

**// Student s = (Student)o;//运行时异常 ClassCastException**

**// s.study();**

**}else{**

**System.out.println("对不起 类型不匹配 不帮您造型啦 否则会出问题");**

**}**

1. **异常**

**ClassCastException 造型异常，将对象的类型还原时，与真实类型不匹配**

**InputMismatchException 输入不匹配**

**NumberFormateException 数字格式化异常**

**ArrayIndexOutOfBoundsException 数组索引越界**

**NegativeArraySizeException 数组长度负数**

**NullPointerException 空指针异常**

**ArithmeticException 算数异常**

**ClassCastException 造型异常 将对象的类型还原时 与真实类型不匹配**

**StackOverflowError 栈内存溢出错误**

1. **策略模式**
2. **内部类**
3. **枚举**
4. **Runtime，GC**

# ****第二章、工具类****

1. **包装类(引用类/封装类🡪Integer/Charcter/Long….)**

**基本类—> int/byte/char..**

**byte---Byte short---Short int---Integer long---Long**

**float---Float double---Double char---Charcter boolean---Boolean**

1. **8个包装类有6个继承Number类，xxxValue(); 🡪将一个包装类类型转化为对应的基本类型(拆包)**

Integer i1 = new Integer(10);//创建引用类型i1  
int value1 = i1.intValue();//将引用类型转换为基本类型，利用了xxxValue();

1. **比较==与equals()的区别：**

**==可以比较基本数据类型和引用数据类型**

**基本数据类型，比较的是变量中储存的值**

**引用数据类型，比较的是变量中储存的地址引用**

**equals()被Integer类重写了，重写后比较的是变量储存的值**

**如果没有重写，则默认继承==方法**

1. **考察Integer类加载的时候 自己有一个静态的空间**

**空间内立即加载 Integer类型的数组 内存储256个Integer对象 -128 ~ 127**

**如果我们用的对象范围在这之内Integer i1 = 10; 直接取静态区中找对应的对象**

**如果我们用的对象范围超出了这个Integer i1 = 1000; 会帮我们创建一个新的Integer对象**

Integer i1 = 1000;  
Integer i2 = 1000;  
Integer i3 = new Integer(1000);  
Integer i4 = new Integer(1000);  
System.*out*.println(i1 == i2);//基本数据类型对比的是值 true， 如果变成1000，这里变成false  
System.*out*.println(i1 == i3);//引用数据类型对比的是地址 false  
System.*out*.println(i3 == i4);//引用数据类型对比的是地址 false  
System.*out*.println(i1.equals(i2));//对比的是值 true  
System.*out*.println(i1.equals(i3));//对比的是值 true  
System.*out*.println(i3.equals(i4));//对比的是值 true

**Integer i1 = 10;是什么意思: 这个是包装类，有两个步骤**

**Integer i1 = new Integer(10);**

**int value = i1.intValue();**

1. **数学相关的类**
2. **Math**

double t = 13.54;  
System.*out*.println(Math.*abs*(-13.54));//绝对值  
System.*out*.println(Math.*ceil*(t));//向上取整 13  
System.*out*.println(Math.*floor*(t));//向下 14  
System.*out*.println(Math.*rint*(t));//临近的 相同取偶数 14  
System.*out*.println(Math.*round*(t));//四舍五入 14

**max(a,b) min(a,b) (参数int long float double)**

**pow(a,b) a的b次方 (参数double 返回值double)**

**sqrt(double a) 获取给定参数的平方根**

**double = random(); 随机产生一个 [0.0--1.0)**

1. **Math.random()**

**0-9之间的随机整数**

**int value = (int)(Math.random()\*10);**

**5.0---10.9之间的小数**

**(Math.random()\*6)+5**

**0.0---0.99999 \* 6**

**(0.0---5.49999)+5**

1. **Random**

**在java.util包中的类 需要import导入**

**没有任何继承关系 默认继承Object类**

**查找构造方法--->如何创建对象**

**Random r = new Random();**

**类中提供的常用方法**

**r.nextInt(); 随机产生 int取值范围的整数 有正有负**

**r.nextInt(int bound); 随机产生一个 [0--bound) 整数**

**注意bound必须为正数 否则会出现如下的运行时异常**

**IllegalArgumentException**

**r.nextFloat() 随机产生一个 [0.0---1.0)**

**r.nextBoolean() 随机产生一个boolean值 true false**