Java核心优势：跨平台

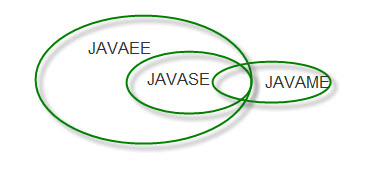
JavaSE(Java Standard Edition)：标准版，定位在个人计算机上的应用。这个版本是Java平台的核心，它提供了非常丰富的API来开发一般个人计算机上的应用程序，包括用户界面接口AWT及Swing，网络功能与国际化、图像处理能力以及输入输出支持等。在上世纪90年代末互联网上大放异彩的Applet也属于这个版本。Applet后来为Flash取代，Flash即将被HTML5取代。

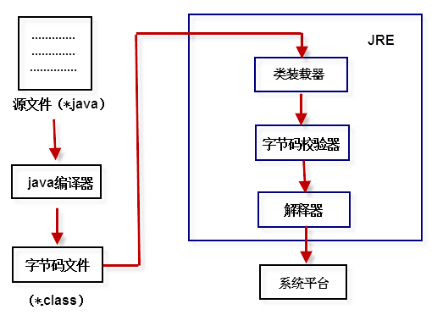
　　JavaEE(Java Enterprise Edition)：企业版，定位在服务器端的应用。

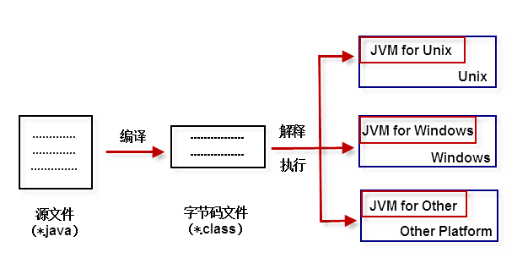
　　JavaEE是JavaSE的扩展，增加了用于服务器开发的类库。如：JDBC是让程序员能直接在Java内使用的SQL的语法来访问数据库内的数据;Servlet能够延伸服务器的功能，通过请求-响应的模式来处理客户端的请求;JSP是一种可以将Java程序代码内嵌在网页内的技术;

　　JavaME(Java Micro Edition)：微型版，定位在消费性电子产品的应用上

JavaME是JavaSE的内伸，包含J2SE的一部分核心类，也有自己的扩展类,增加了适合微小装置的类库：javax.microedition.io.\*等。该版本针对资源有限的电子消费产品的需求精简核心类库，并提供了模块化的架构让不同类型产品能够随时增加支持的能力。

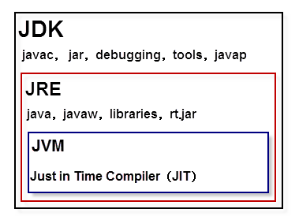






Java Runtime Environment (JRE) 包含：Java虚拟机、库函数、运行Java应用程序所必须的文件。

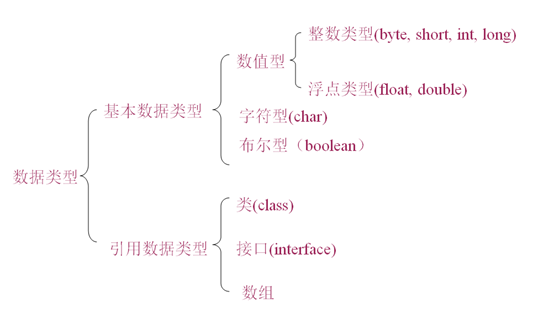
Java Development Kit (JDK)包含：包含JRE，以及增加编译器和调试器等用于程序开发的文件。



|  |  |  |  |
| --- | --- | --- | --- |
| **表2-2局部变量、成员变量、静态变量的区别** | | | |
| 类型 | 声明位置 | 从属于 | 生命周期 |
| 局部变量 | 方法或语句块内部 | 方法/语句块 | 从声明位置开始，直到方法或语句块执行完毕，局部变量消失 |
| 成员变量  (实例变量) | 类内部，方法外部 | 对象 | 对象创建，成员变量也跟着创建。对象消失，成员变量也跟着消失； |
| 静态变量  (类变量) | 类内部，static修饰 | 类 | 类被加载，静态变量就有效；类被卸载，静态变量消失。 |

变量和常量命名规范：

1. 所有变量、方法、类名：见名知意
2. 类成员变量：首字母小写和驼峰原则:  monthSalary
3. 局部变量：首字母小写和驼峰原则
4. 常量：大写字母和下划线：MAX\_VALUE
5. 类名：首字母大写和驼峰原则:  Man, GoodMan
6. 方法名：首字母小写和驼峰原则: run(), runRun()



4 bytes

2 bytes

1 bit

1 2 4 8 bytes

4 8 bytes

注：引用数据类型的大小统一为4个字节，记录的是其引用对象的地址！

|  |  |  |
| --- | --- | --- |
| **表2-4整型数据类型** | | |
| **类型** | **占用存储空间** | **表数范围** |
| byte | 1字节 | -27~   27-1（-128~127） |
| short | 2字节 | -215~   215-1（-32768~32767） |
| int | 4字节 | -231~   231-1 (-2147483648~2147483647)约21亿 |
| long | 8字节 | -263~   263-1 |

Java 语言整型常量的四种表示形式

十进制整数，如：99, -500, 0

八进制整数，要求以 0 开头，如：015

十六进制数，要求 0x 或 0X 开头，如：0x15

二进制数，要求0b或0B开头，如：0b01110011

Java语言的整型常数默认为int型，声明long型常量可以后加‘ l ’或‘ L ’ 。

|  |  |  |
| --- | --- | --- |
| **表2-5浮点型数据类型** | | |
| **类型** | **占用存储空间** | **表数范围** |
| float | 4字节 | -3.403E38~3.403E38 |
| double | 8字节 | -1.798E308~1.798E308 |

E表示科学计数法以10为底数

float类型又被称作单精度类型，尾数可以精确到7位有效数字，在很多情况下，float类型的精度很难满足需求。而double表示这种类型的数值精度约是float类型的两倍，又被称作双精度类型，绝大部分应用程序都采用double类型。浮点型常量默认类型也是double。

float类型赋值时需要添加后缀F/f：

float f=3.14F; //小数默认是double类型，double转float会丢失精度因此需要加f/F

浮点数是不精确的，一定不要用于比较(若一定需要比较)

java.math包下面的两个有用的类：BigInteger和BigDecimal，这两个类可以处理任意长度的数值。BigInteger实现了任意精度的整数运算。BigDecimal实现了任意精度的浮点运算。

eg:

BigDecimal bd = BigDecimal.valueOf(1.0);

bd = bd.subtract(BigDecimal.valueOf(0.1));//用BigDecimal方法-0.1

bd = bd.subtract(BigDecimal.valueOf(0.1));

bd = bd.subtract(BigDecimal.valueOf(0.1));

bd = bd.subtract(BigDecimal.valueOf(0.1));

bd = bd.subtract(BigDecimal.valueOf(0.1));

System.out.println(bd);//0.5

System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);//0.5000000000000001(直接-0.1)

BigDecimal bd2=BigDecimal.valueOf(0.1);

BigDecimal bd3=BigDecimal.valueOf(1.0/10.0);

System.out.println(bd2.equals(bd3));//true

System.out.println('a'+'b');//195 字符间出现加号则字符自动转为ascall码

System.out.println(""+'a'+'b');//ab 前加空字符串变成输出字符

System.out.println(""+'a'+'\n'+'b');//换行

System.out.println(""+'a'+'\t'+'b');//tab

System.out.println(""+'a'+'\''+'b');//a'b

boolean类型有两个常量值，true和false，在内存中占一位（不是一个字节），不可以使用 0 或非 0 的整数替代 true 和 false ，这点和C语言不同。

注意：java中true/false不能用整数替代!

老鸟的写法是if ( flag )或者if ( !flag) ==容易写成=

|  |  |  |
| --- | --- | --- |
| 算术运算符 | 二元运算符 | +，-，\*，/，% |
| 一元运算符 | ++，-- |
| 赋值运算符 | | = |
| 扩展运算符 | | +=，-=，\*=，/= |
| 关系运算符 | | >，<，>=，<=，==，!=  instanceof |
| 逻辑运算符 | | &&，||，!，^ |
| 位运算符 | | &，|，^，~ ， >>，<<，>>> |
| 条件运算符 | | ? : |
| 字符串连接符 | | + |

算术运算符：

byte a=1;

int b=2;

long b2=3;

//byte c=a+b; 不能将byte转为int

//int c2=b2+b; 不能将int转为long

float f1=3.14f;

float d=b+b2;

//float d2=f1+6.2; 不能将double转为float

//取余余数符号和左边操作数相同

System.out.println(9%5);//4

System.out.println(-9%5);//-4

int a1 = 3;

int b1 = a1++; //执行完后,b=3。先给b赋值，再自增。

System.out.println("a1="+a1+"\nb1="+b1);

a1 = 3;

b1 = ++a1; //执行完后,b=4。a先自增，再给b赋值

System.out.println("a1="+a1+"\nb1="+b1);

赋值运算符：

a=3;

b=4;

a+=b;//相当于a=a+b;

System.out.println("a="+a+"\nb="+b);

a=3;

a\*=b+3;//相当于a=a\*(b+3)

System.out.println("a="+a+"\nb="+b);

|  |  |  |
| --- | --- | --- |
| 运算符 | | 说明 |
| 逻辑与 | &( 与) | 两个操作数为true，结果才是true，否则是false |
| 逻辑或 | |(或) | 两个操作数有一个是true，结果就是true |
| 短路与 | &&( 与) | 只要有一个为false，则直接返回false |
| 短路或 | ||(或) | 只要有一个为true， 则直接返回true |
| 逻辑非 | !（非） | 取反：!false为true，!true为false |
| 逻辑异或 | ^（异或） | 相同为false，不同为true |

短路与和短路或采用短路的方式。从左到右计算，如果只通过运算符左边的操作数就能够确定该逻辑表达式的值，则不会继续计算运算符右边的操作数，提高效率。

//1>2的结果为false，那么整个表达式的结果即为false，将不再计算2>(3/0)

boolean c = 1>2 && 2>(3/0);

System.out.println(c);

//1>2的结果为false，那么整个表达式的结果即为false，还要计算2>(3/0)，0不能做除数，//会输出异常信息

boolean d = 1>2 & 2>(3/0);

System.out.println(d);

|  |  |
| --- | --- |
| 位运算符 | 说明 |
| ~ | 取反 |
| & | 按位与 |
| | | 按位或 |
| ^ | 按位异或 |
| << | 左移运算符，左移1位相当于乘2 |
| >> | 右移运算符，右移1位相当于除2取商 |

左移右移运算效率高

int a = 3\*2\*2;

int b = 3<<2; //相当于：3\*2\*2;

int c = 12/2/2;

int d = 12>>2; //相当于12/2/2;

雷区：

　　1. &和|既是逻辑运算符，也是位运算符。如果两侧操作数都是boolean类型，就作为逻辑运算符。如果两侧的操作数是整数类型，就是位运算符。

　　2. 不要把“^”当做数学运算“乘方”，是“位的异或”操作。

字符串连接符：

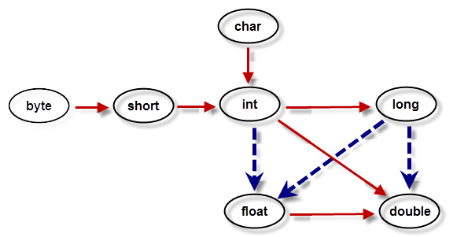
“+”运算符两侧的操作数中只要有一个是字符串(String)类型，系统会自动将另一个操作数转换为字符串然后再进行连接。(因此想输出字符串，在最前面加空字符串则后面加法均为字符串连接符)

|  |  |  |  |
| --- | --- | --- | --- |
| 优先级 | 运算符 | 类 | 结合性 |
| 1 | () | 括号运算符 | 由左至右 |
| 2 | !、+（正号）、-（负号） | 一元运算符 | 由左至右 |
| 2 | ~ | 位逻辑运算符 | 由右至左 |
| 2 | ++、-- | 递增与递减运算符 | 由右至左 |
| 3 | \*、/、% | 算术运算符 | 由左至右 |
| 4 | +、- | 算术运算符 | 由左至右 |
| 5 | <<、>> | 位左移、右移运算符 | 由左至右 |
| 6 | >、>=、<、<= | 关系运算符 | 由左至右 |
| 7 | ==、!= | 关系运算符 | 由左至右 |
| 8 | & | 位运算符、逻辑运算符 | 由左至右 |
| 9 | ^ | 位运算符、逻辑运算符 | 由左至右 |
| 10 | | | 位运算符、逻辑运算符 | 由左至右 |
| 11 | && | 逻辑运算符 | 由左至右 |
| 12 | || | 逻辑运算符 | 由左至右 |
| 13 | ? : | 条件运算符 | 由右至左 |
| 14 | =、+=、-=、\*=、/=、%= | 赋值运算符、扩展运算符 | 由右至左 |

逻辑与、逻辑或、逻辑非的优先级一定要熟悉！（逻辑非>逻辑与>逻辑或）。

如：a||b&&c的运算结果是：a||(b&&c)，而不是(a||b)&&c

自动类型转换指的是容量(表示范围，而不是占用字节数)小的数据类型可以自动转换为容量大的数据类型。如图2-6所示，黑色的实线表示无数据丢失的自动类型转换，而虚线表示在转换时可能会有精度的损失。



可以将整型常量直接赋值给byte、 short、 char等类型变量，而不需要进行强制类型转换，只要不超出其表数范围即可。(特例)

short b = 12; //合法 注意：float a=3.6;不合法因为double不能自动转换为float

short b = 1234567;//非法，1234567超出了short的表数范围

溢出问题：

int money = 1000000000; //10亿

int years = 20;

//返回的total是负数，超过了int的范围

int total = money\*years;

System.out.println("total="+total);

//返回的total仍然是负数。默认是int，因此结果会转成int值，再转成long。但是已//经发生了数据丢失

long total1 = money\*years;

System.out.println("total1="+total1);

//返回的total2正确:先将一个因子变成long，整个表达式发生提升。全部用long来计算。

long total2 = money\*((long)years);

System.out.println("total2="+total2);

使用Scanner获取键盘输入：

Scanner scanner = new Scanner(System.in);

System.out.println("请输入名字：");

String name = scanner.nextLine();

switch语句：

char c = 'a';

int rand = (int) (26 \* Math.random());

char c2 = (char) (c + rand);

System.out.print(c2 + ": ");

switch (c2) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

System.out.println("元音");

break;

case 'y':

case 'w':

System.out.println("半元音");

break;

default:

System.out.println("辅音");

带标签的break和countinue:

对Java来说唯一用到标签的地方是在循环语句之前。而在循环之前设置标签的唯一理由是：我们希望在其中嵌套另一个循环，由于break和continue关键字通常只中断当前循环，但若随同标签使用，它们就会中断到存在标签的地方。

eg:break和continue：控制嵌套循环跳转(打印101-150之间所有的质数)

outer: for (int i = 101; i < 150; i++) {

for (int j = 2; j < i / 2; j++) {

if (i % j == 0){

continue outer;//找到除数就不用再向后继续找(从内循环跳到外循环)

}

}

System.out.print(i + " ");

}

方法的详细说明

1. 形式参数：在方法声明时用于接收外界传入的数据。

2. 实参：调用方法时实际传给方法的数据。

3. 返回值：方法在执行完毕后返还给调用它的环境的数据。

4. 返回值类型：事先约定的返回值的数据类型，如无返回值，必须显示指定为void。

return 语句终止方法的运行并指定要返回的数据。

Java中进行方法调用中传递参数时，遵循值传递的原则(传递的都是数据的副本)。

方法的重载是指一个类中可以定义多个方法名相同，但参数不同的方法。调用时，会根据不同的参数自动匹配对应的方法。

雷区

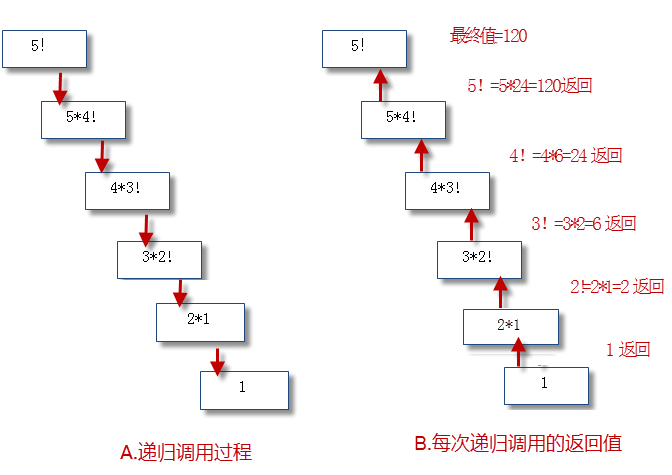
重载的方法，实际是完全不同的方法，只是名称相同而已!

构成方法重载的条件：

1.不同的含义：形参类型、形参个数、形参顺序不同。

2.只有返回值不同不构成方法的重载。(无法区分)

3.只有形参的名称不同，不构成方法的重载。(无法区分)



递归结构包括两个部分：

1.定义递归头。解答：什么时候不调用自身方法。如果没有头，将陷入死循环，也就是递归的结束条件。

2.递归体。解答：什么时候需要调用自身方法。

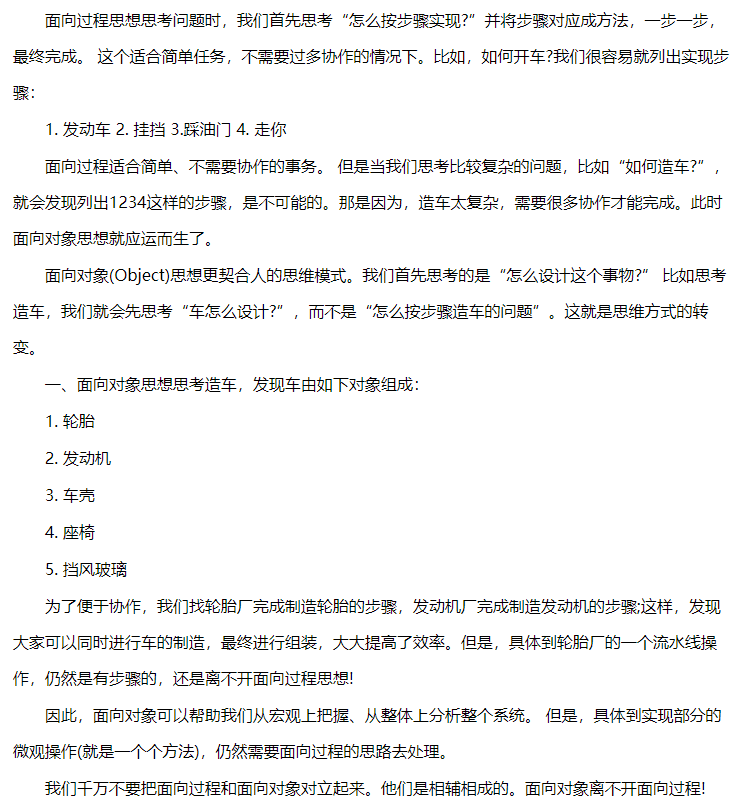
递归的缺陷

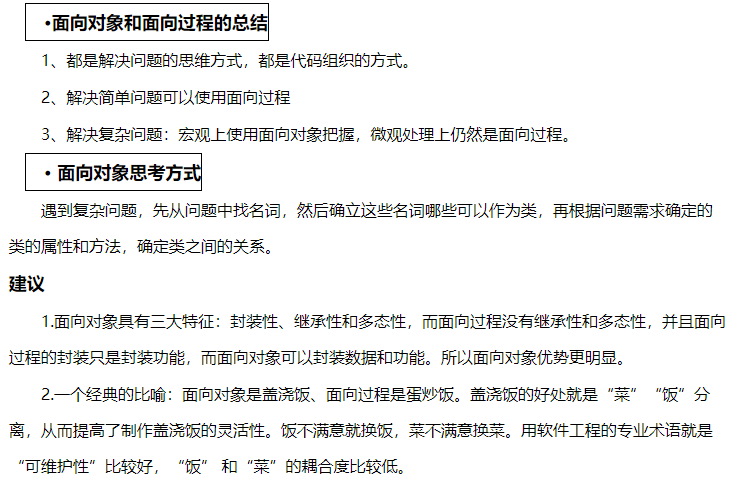
简单的程序是递归的优点之一。但是递归调用会占用大量的系统堆栈，内存耗用多，在递归调用层次多时速度要比循环慢的多，所以在使用递归时要慎重。

注意事项

任何能用递归解决的问题也能使用迭代解决。当递归方法可以更加自然地反映问题，并且易于理解和调试，并且不强调效率问题时，可以采用递归;

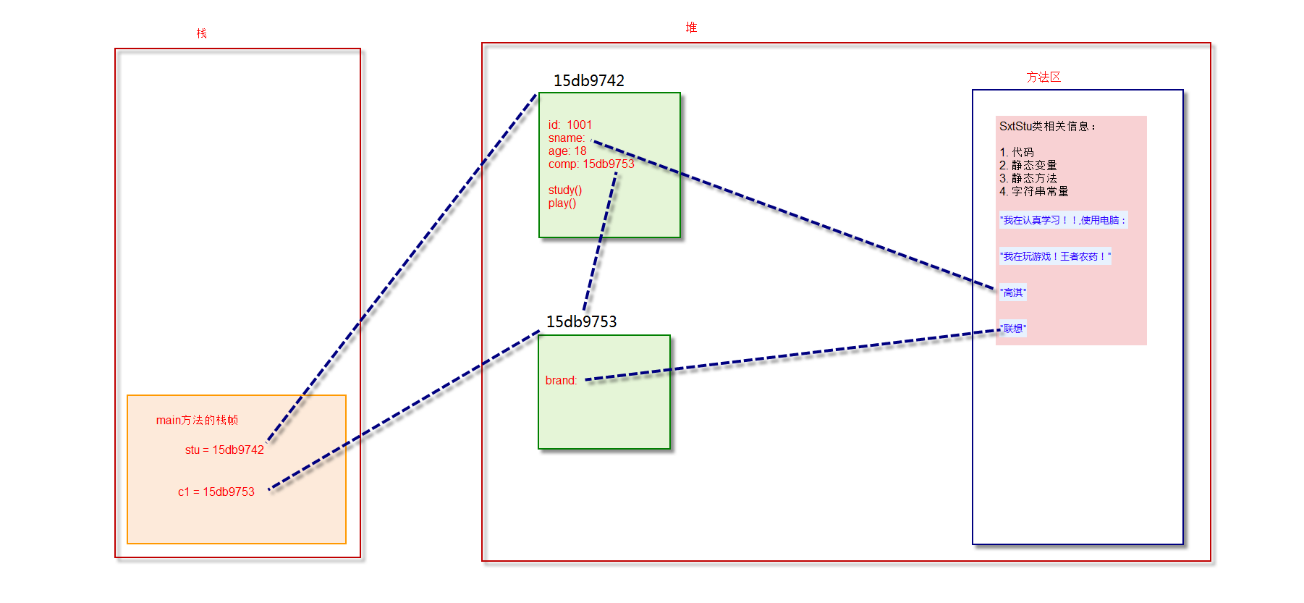
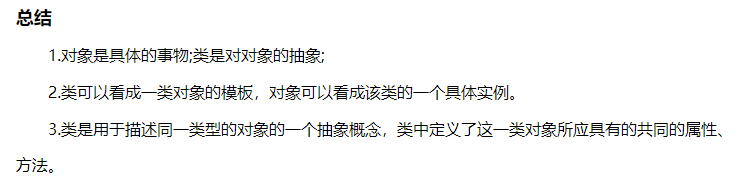
在要求高性能的情况下尽量避免使用递归，递归调用既花时间又耗内存。

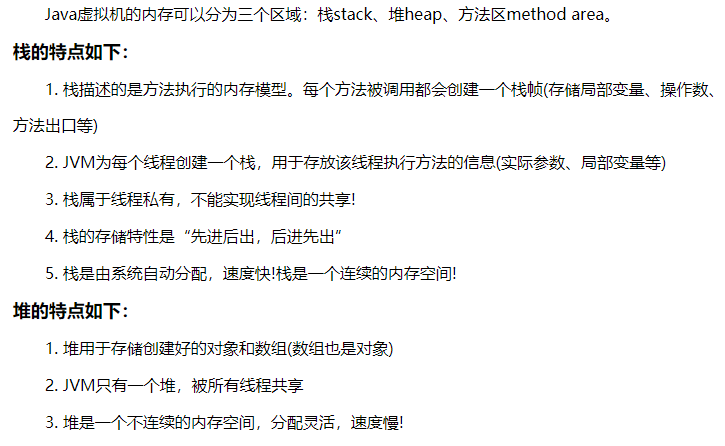


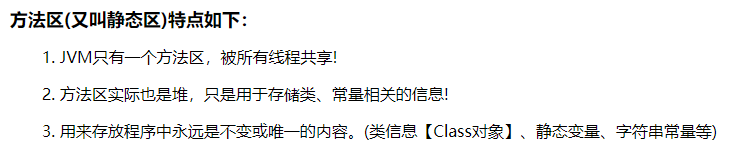
·

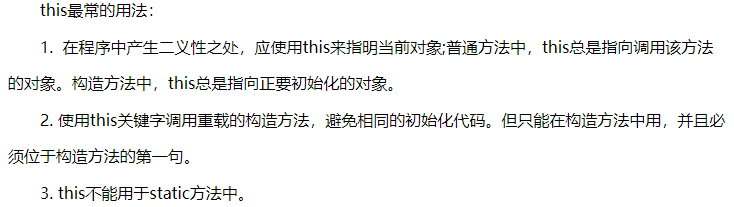
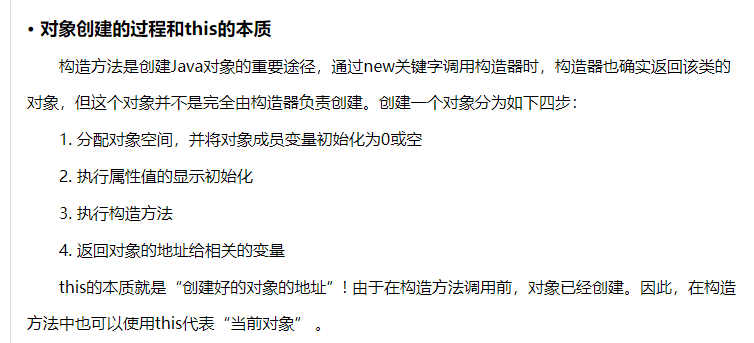
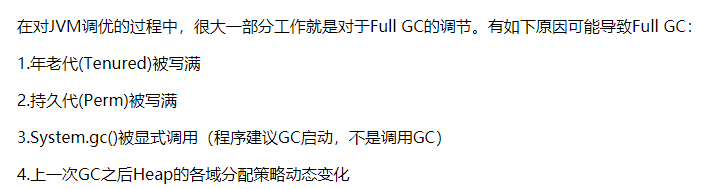
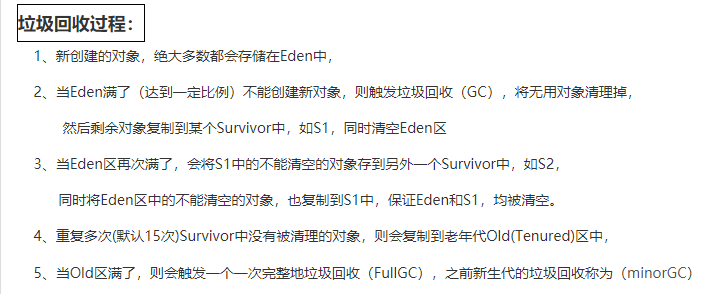
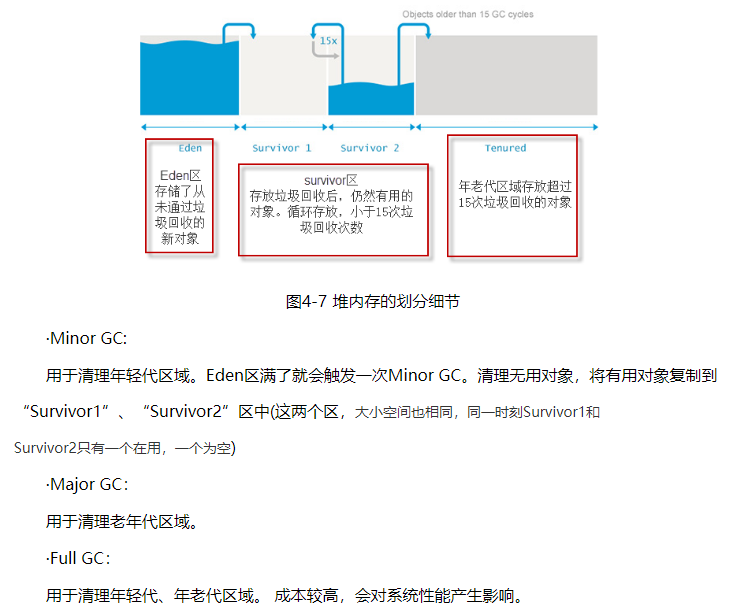
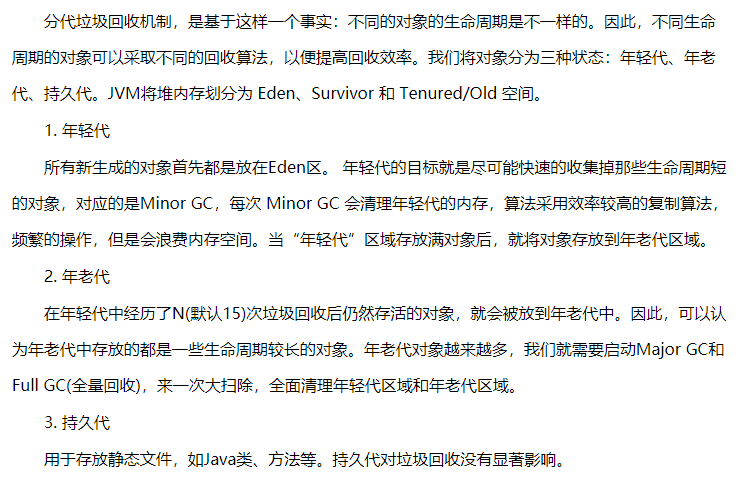
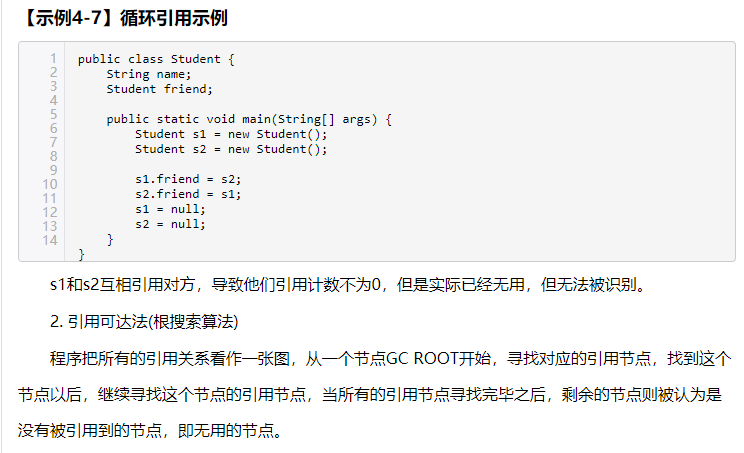
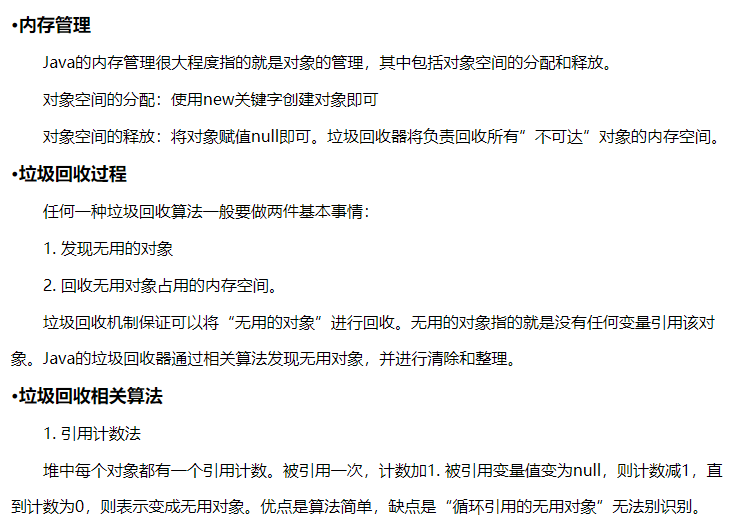
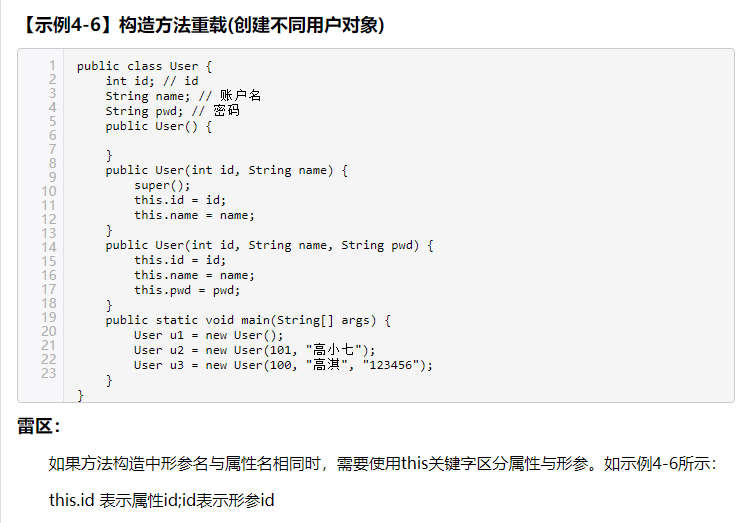
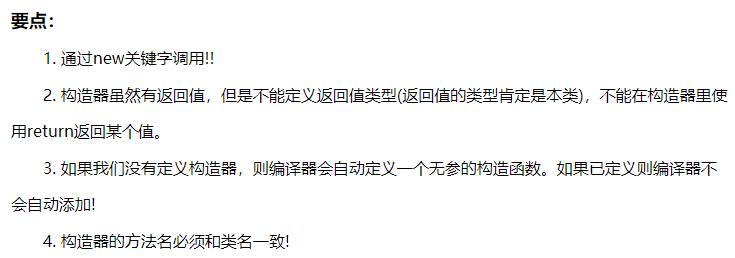
数据->数组->对象

无管理->企业部门制(同一类型放入一个部门)->企业项目制(每个项目组有不同类型)





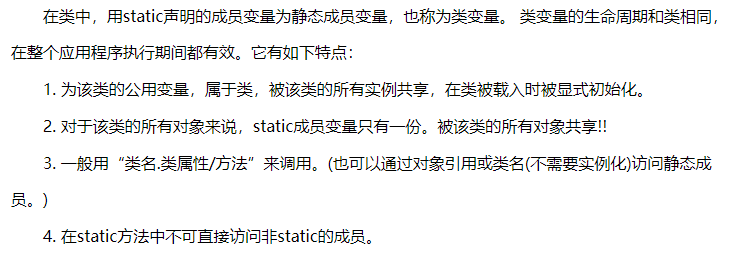
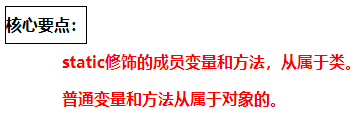
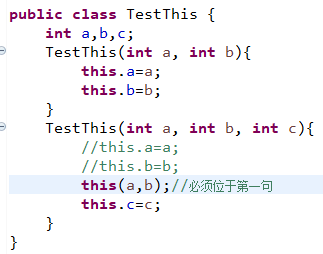


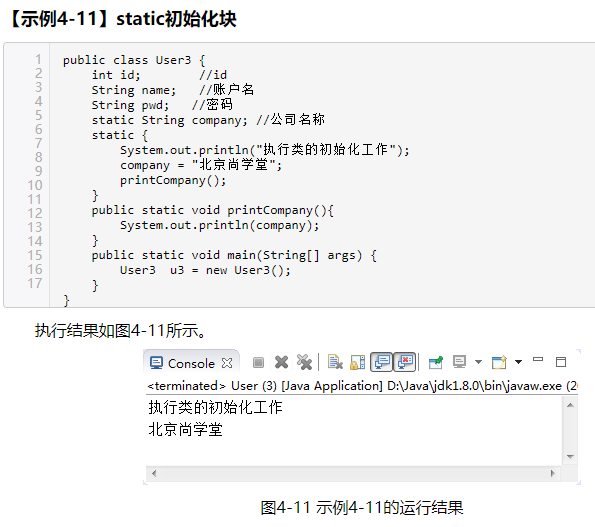
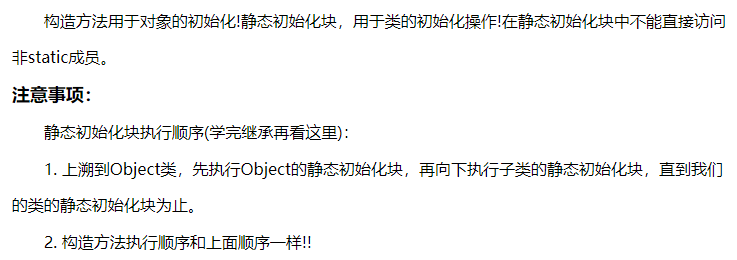
构造方法：  
 

2

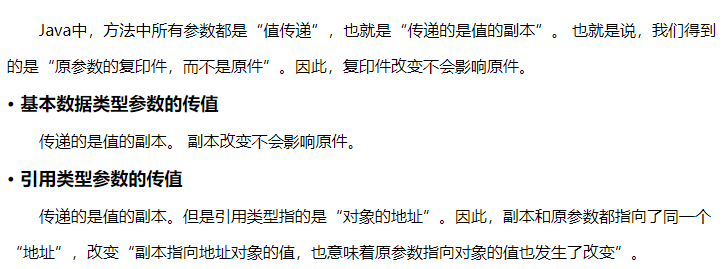
式子

静态方法位于方法区中，不是对象，不能使用this和对象中的方法。

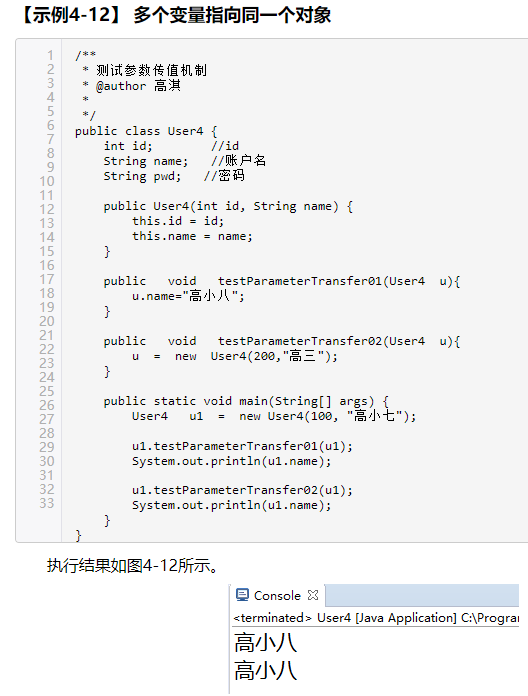


方法区相当于图纸，对象相当于汽车，汽车可以调用图纸中的内容，图纸无法调用某一汽车的内容，方法区和类的生命周期相同，因此类创建则方法区创建，但不一定创建对象。

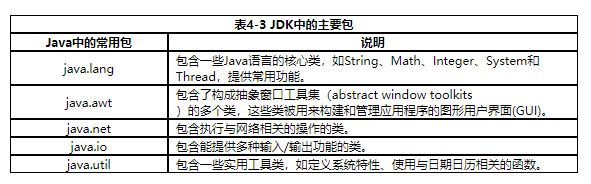
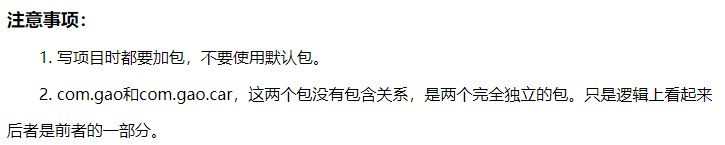
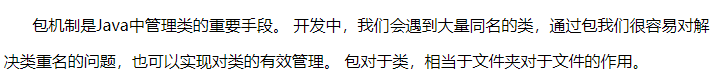
先加载类，再加载构造器。



传递对象的值是传递对象的地址，虽然也是值传递，但会改变原对象地址指向的值。

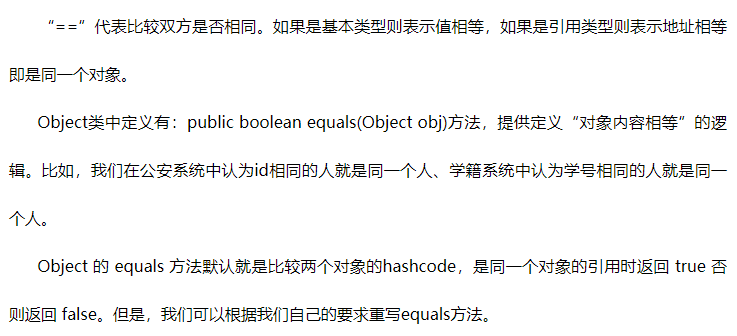
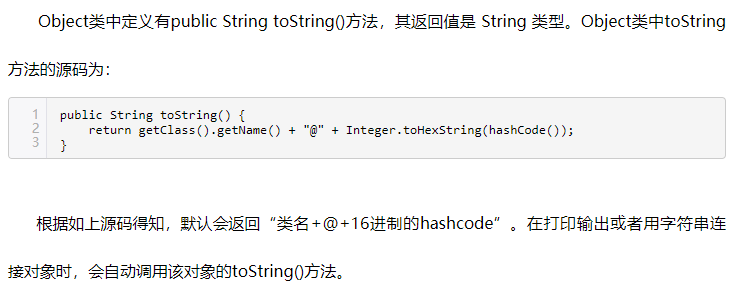
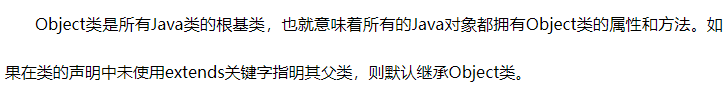
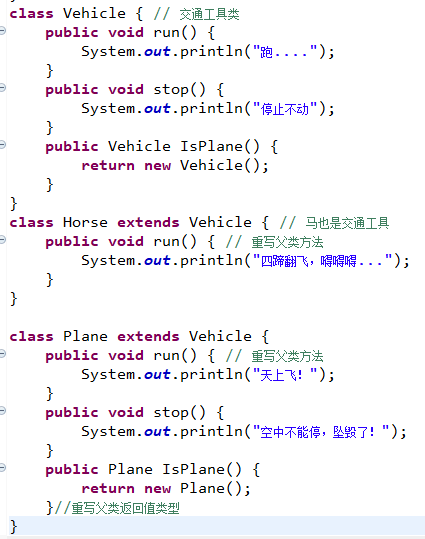
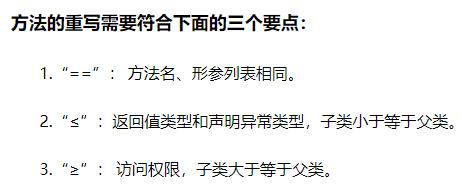
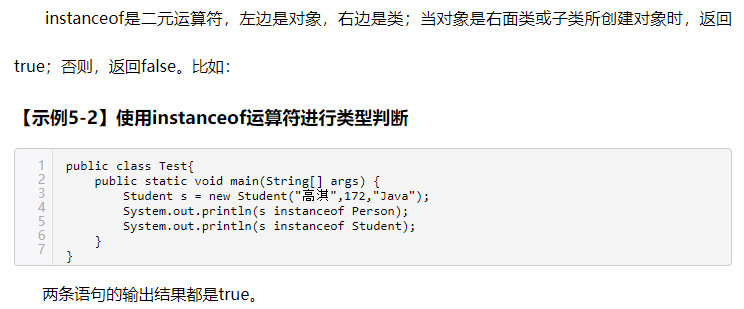
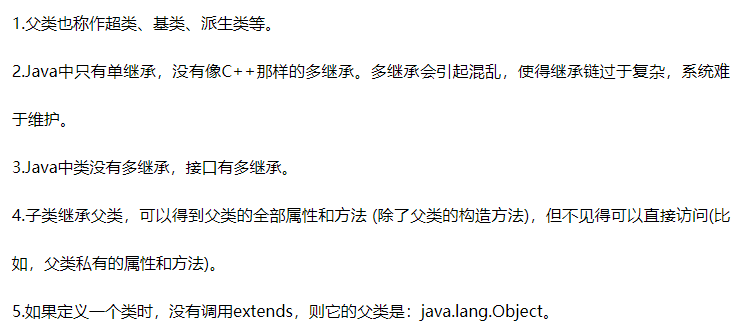


第二个输出值没有变的原因是u1地址指向的对象没有发生改变，而是新建的另一个地址的对象发生了改变。

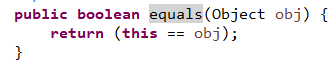


java.lang属于语言包，不用导入直接可以使用。

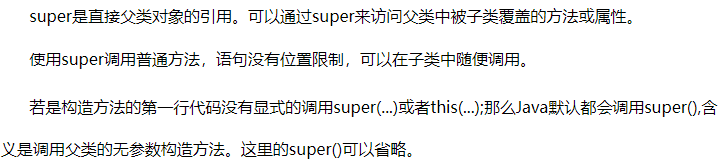
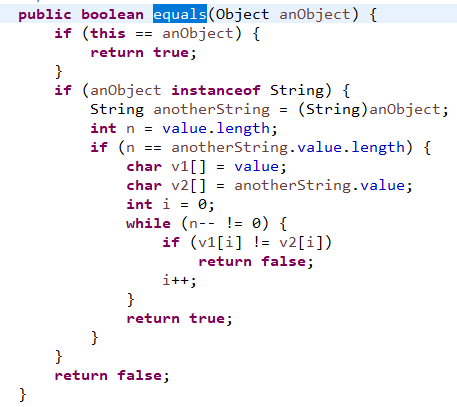




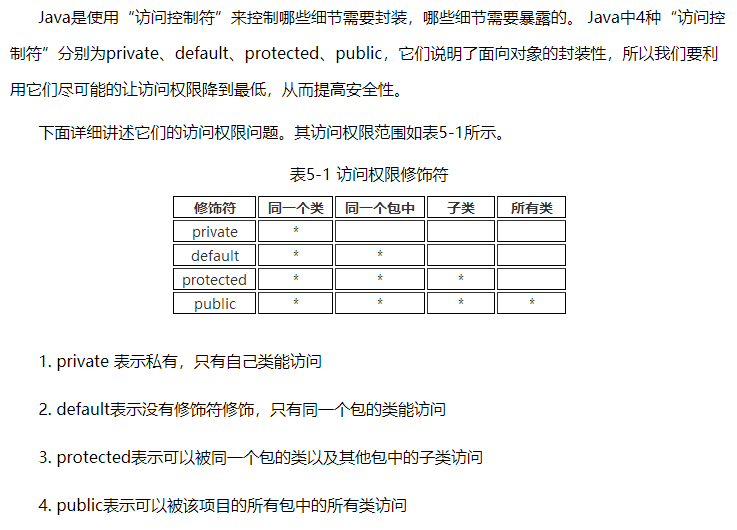
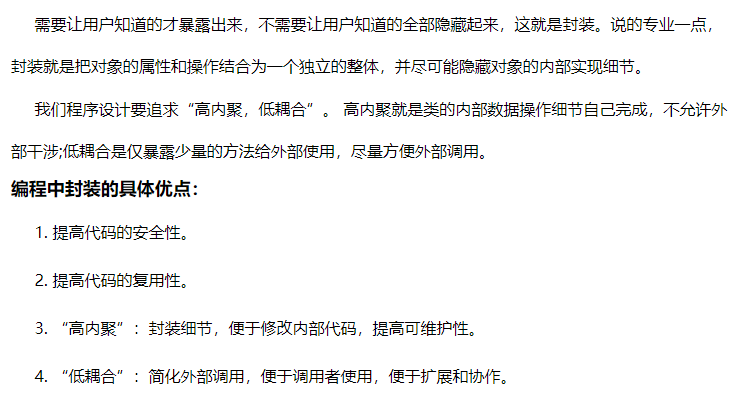
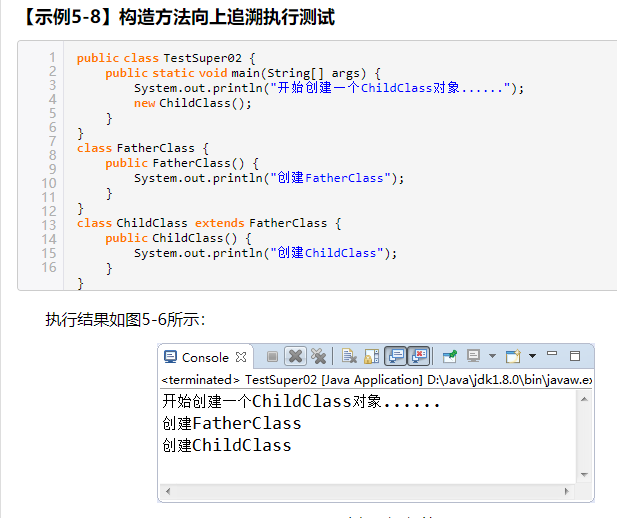
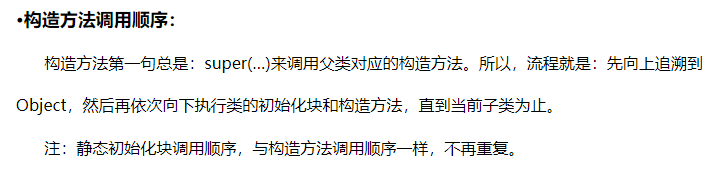
Object的equals方法：

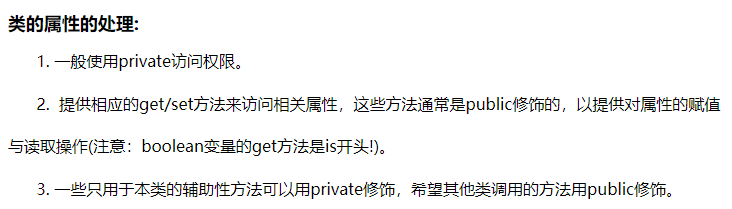


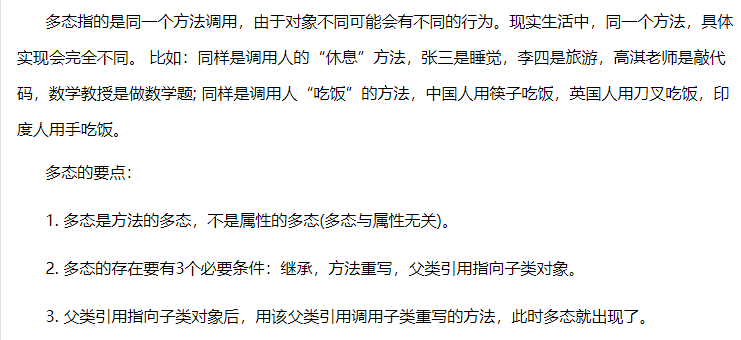
String的equals方法(对字符串中每个字符进行比较)：如果是==则必须是同一对象



用super.父类方法/父类成员变量 来调用父类的方法/成员变量



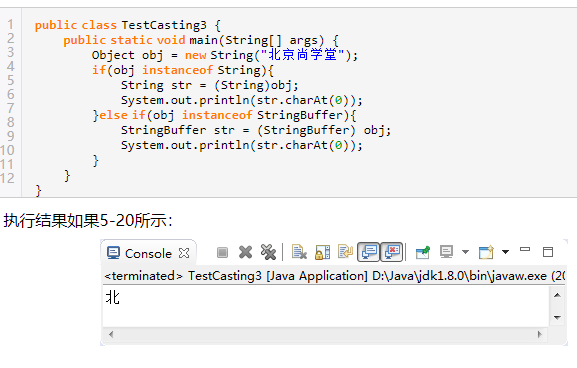


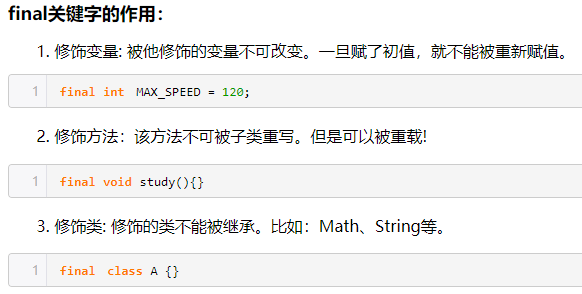


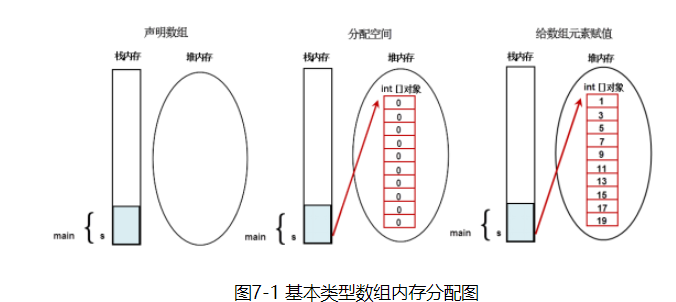
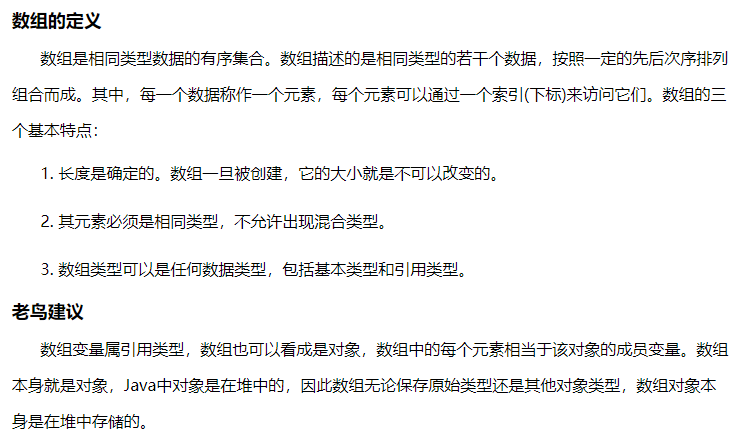


注意：jvm可以认识a2是Dog类，但编译器认为a2就是Animal类，没有seeDoor方法，因此编译器会报错，通不过编译器的检查。(将Dog类强转为Cat类可以通过编译器的检查，但通不过jvm运行检查)

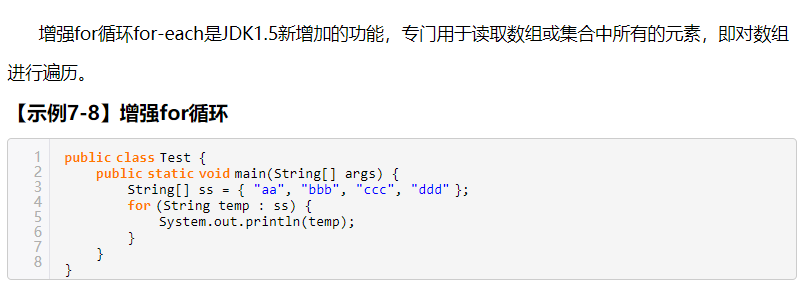
向下转型可以用instanceof运算符进行判断







分配空间相当于创建数组对象，给数组元素赋值相当于给数组对象成员变量赋值。



只能读取不能修改(没有下标)