

<#>

请到场的同学扫码签到



Java A互助课

堂

lecture2

导生：杨宗奇

- 互助课堂仅作为老师讲课内容的解释和补充，请不要以听互助课堂取代听大课
- 本课件主要内容来自于老师上课课件，并对其作了修改和补充。
- 互助课堂旨在帮助大家提升学业成绩，有困难之处欢迎讨论交流，但要**杜绝抄袭、代写等行为。不要以可存留的方式分享代码**
- 课件中标红或有 ★ 标志的个人认为是需要重点理解的内容。有 ▲ 标志的表示需要记忆但不用理解很深的内容。

目录

CONTENTS

01

上节回顾

02

分支结构

03

循环结构

04

数组

05

如何进行调试

06

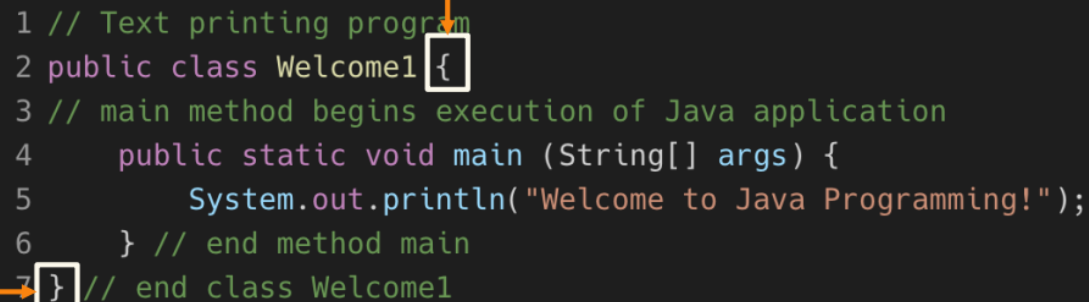
课后练习

上节回顾

- 冯诺依曼结构：I O M A C S
- Java是一种半解释半编译的语言
- JDK=JRE + Development tools, JRE=JVM + Library Classes
- 三种程序错误类型：Syntax(语法)、Runtime(运行)、Semantic(语义)
- System所在的库为**java.lang**，该库默认被程序引用

- The braces

- A **left brace**, {, begins the body of every class declaration
- A corresponding **right brace**, }, must end each class declaration
- Code between braces should be indented (good practice)



```
1 // Text printing program
2 public class Welcome1 {
3 // main method begins execution of Java application
4     public static void main (String[] args) {
5         System.out.println("Welcome to Java Programming!");
6     } // end method main
7 } // end class Welcome1
```

- ★ • **作用域(scoop):**花括号表示了java中的作用域，在某个作用域中产生的东西只能在其中使用。



- 赋值时从右往左
- 变量会被新赋的值给覆盖

练习：若有相同类型的变量a, b,如何用第三个变量t实现a, b值的交换？

```
int a=1,b=2,t;  
t=a;  
a=b;  
b=t;  
System.out.printf("a:%d b:%d",a,b);
```

```
a:2 b:1  
Process finished with exit code 0
```




- 在java中，小数默认为double类型，整数默认为int类型
- 同一个类型的值进行运算，只能得到该类型值的结果。
- 两个不同类型的值进行运算，会将精度低的自动转变为精度高的类型，再进行运算
- 高转低可以强制转换，会失去精度。方法为在要转换的变量前加括号，括号内写转换类型

(double)

- Primitive type精度排序（由低到高）
- (char, byte, short) < int < long < float < double
- char, byte, short 计算前都先转化为int

<#>

复合赋值运算符



- *, /, %, +, -都可以写成如下的表达式

variable operator= expression;

变量名 \longrightarrow 第 c $\xrightarrow{\text{运算符}}$ += 3;

c = c + 3;

<#>

自增自减运算



- 单独成句时，`a++;` 和 `++a;` 都表示变量 `a` 的值+1
- 但当要使用到 `a` 变量时，两者略有不同

```
int a=1,b=0;  
b=a++;  
System.out.println(a);  
System.out.println(b);
```

等价



```
int a=1,b=0;  
b=a;  
a=a+1;  
System.out.println(a);  
System.out.println(b);
```



a: 2
b: 1

```
int a=1,b=0;  
b=++a;  
System.out.println(a);  
System.out.println(b);
```

等价



```
int a=1,b=0;  
a=a+1;  
b=a;  
System.out.println(a);  
System.out.println(b);
```



a: 2
b: 2

<#>

判等运算符

- == 用于判断两个相同类型的值是否相等，结果为一个布尔类型的值，若相等则表达式的值为true，**注意与赋值符=区分**
- != 表示不等于，若两个值不相等则表达式的值为true
- 不应用这类运算符直接比较浮点数

```
double f1=1.1f;  
double f2=110.0/100;  
System.out.println(f1==f2);
```

false

Process finished with exit code 0

直接判命题F/T



&& (conditional AND)

|| (conditional OR)

& (boolean logical AND)

| (boolean logical inclusive OR)

^ (boolean logical exclusive OR)

! (logical NOT)

- &&, ||, !, 只用来表达布尔类型值之间的关系
- &, |, ^ 可以作用于整型的每个二进制位
- && 有0出0, 全1出1
- || 有1出1, 全0出0

- 涉及 `&&` 或 `||` 的表达式中，若在运算符左边的表达式已经能确定表达式的值，则不会对其余的表达式进行计算。

```
boolean b;  
int a=1,c=1;  
b=(a==1 || a==c);  
b=(a==0 && a==c);
```

- 以下两个表达式都不会执行对 `a==c` 的判断，因为第一个表达式已经可以确定这个运算的结果

优先级	运算符
1	()
2	! +(正) -(负) ++ --
3	* / %
4	+(加) -(减)
5	< <= > >=
6	== !=
7	^
8	&&
9	
10	?:
11	= += -= *= /= %=

• 括号最优先

• 算术运算

• 结果为逻辑值的运算

• 对逻辑值的运算（非>与>或）

• 赋值运算

<#>

复合运算

```
boolean b=false;
int a=1,c=2;
if(b=true || ++a==c++)
    System.out.println(a+c);
else
    System.out.println(a-c);
```

Handwritten annotations:
- Blue '2 2' above the second '2' in `c=2`.
- Red arrow pointing to `b=true` with the red text "赋值" (Assignment).

• 请分析一下运算的顺序

以上代码输出结果为

D

A.1 B.-1 C.5 D.3

The image features a light gray background with decorative geometric elements in the corners. In the top-left corner, there are two overlapping parallelograms: a dark blue one in the foreground and a light blue one behind it. In the bottom-right corner, there are also two overlapping parallelograms: a dark blue one in the foreground and a light blue one behind it. The text '分支结构' is centered in the middle of the page.

分支结构

<#>

if-else



```
if(逻辑表达式)
```

```
    Do A
```

```
else
```

```
    Do B
```

```
if (x > 5)
    if (y > 5)
        System.out.println("x and y are > 5");
else
    System.out.println("x is <= 5");
```

```
if (x > 5) {
    if (y > 5)
        System.out.println("x and y are > 5");
} else
    System.out.println("x is <= 5");
```

- if和else下面的语句都是一个作用域，单句不用加花括号，多句要加

- if-else可以嵌套，else自动和其同一作用域内最近的一个if配对

<#>

三目运算符

```
if(逻辑表达式)
```

```
    Do A
```

```
else
```

```
    Do B
```

```
(逻辑表达式) ? Do A : Do B;
```

xx ? Do A : Do B;

- 三目运算符?: 是对两元if-else的简写，表达式为true则执行第一个位置的操作，为false则执行第二个位置的操作。

<#>

switch



```
switch(变量名)
{
    case 值1: Do A
    break;
    case 值2: Do B
    break;
    .....
    default: Do something
}
```

找起点

- case后面只能填该变量类型的具体值，不能填逻辑表达式
- 若变量名不等于case中的任何一个值则运行default后面的内容
- switch本质上是利用变量的值在各个case和default中找到一个程序执行的起点。起点找到后无需再考虑case的值，直接从上往下执行语句，遇到break则退出该作用域

```
1 if (x == 1) {  
2     ;  
3 } else if (x == 2) {  
4     ;  
5 } else {  
6     ;  
7 }
```

- 所有语句都以分号；表示结束。
- 若要表达什么都不做，则直接分号即可

The image features a light gray background with decorative geometric elements in the corners. In the top-left corner, there are overlapping parallelogram shapes in a dark blue-gray and a light teal color. In the bottom-right corner, there are similar overlapping parallelogram shapes in the same dark blue-gray and light teal colors.

循环结构

<#>

while循环



```
while(逻辑表达式)
{
    Do something
}
```

```
i=1;
while(i<=10)
{
    sum+=i;
    i++;
}
```

- 要有一个控制循环出口的变量，该变量应该**初始化**并且在循环体中**其值应改变**
- 一般应用于不确定具体循环次数，但知道什么情况下需要退出循环的算法。例如读到0就停止读入

<#>

do-while循环

```
do
{
    Do something
}
while(逻辑表达式)
```

```
Do something
while(逻辑表达式)
{
    Do something
}
```

- 至少做一次，然后再判断。例如，先吃一口苹果才能知道苹果甜不甜，要不要继续吃。
- 所有do-while都可以翻译成while，因此一般只在需要这样的表达并且while循环体内做的事情比较多时使用。

<#>

for循环



```
for(初始化;循环条件;循环一次后的操作)
{
    Do something
}
```

```
for(int i=1;i<=n;i++)
{
    Do something
}
```

```
int i=1;
while(i<=n)
{
    Do something
    i++;
}
```

- for循环是while的一种改写，一般用于知道循环次数的情况。
- for后面括号被分号分为3个部分，通常分别为初始化、循环执行条件，以及每次循环后循环变量的变化。
- 循环中我们往往会对**循环变量**进行利用，用来简化代码。

<#>

for循环

- 以上只是for循环的一般写法，但有时为了简化代码，for循环的写法可以更灵活。每个被分号划分的区域都可以写多个语句，用逗号隔开即可。

```
for(int i=1;i<=n && i>=m;i*=2)
{
    Do something
}
```

```
for(int l=1,r=n;l<=r;l++,r--)
{
    Do something
}
```

<#>

for循环

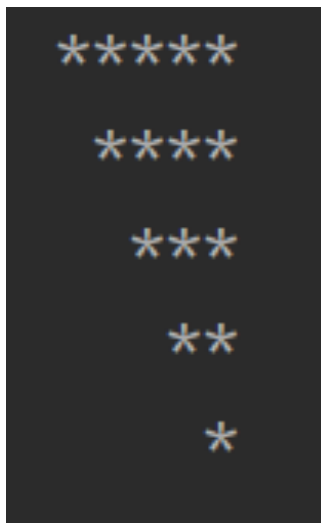
- for括号内定义的变量也属于for的作用域，出了for作用域将无法访问到该变量（除非外面还有定义）

```
int n=10, sum=0;  
for(int i=1; i<=n; i++)  
{  
    sum+=i;  
}  
System.out.println(i);
```

java: 找不到符号
符号: 变量 i
位置: 类 test

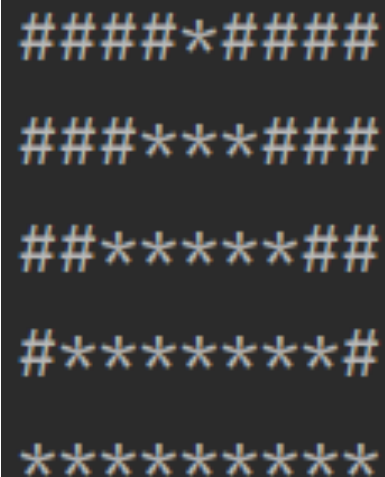
- ★ • 理解循环嵌套要把实际问题分布，并与循环层数对应。
- 请理解以下代码并画出其输出的图形

```
int n=5;
for(int i=1;i<=n;i++)//第几层
{
    //三个同级语句，因此每层做了三件事
    for(int j=1;j<=i-1;j++)
        System.out.printf(" ");
    //第一件是输出i-1个空格
    for(int j=1;j<=n-i+1;j++)
        System.out.printf("*");
    //第二件是输出n-i+1个*
    System.out.printf("\n");
    //第三件是换行
}
```



```
*****
 ****
  ***
   **
    *
```

- 请利用循环画出如下图形(n=5)



```
#####*#####
###***#####
##*****##
#*****#
*****
```

```
int n=5;
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=n-i;j++)
        System.out.printf("#");
    for(int j=1;j<=2*i-1;j++)
        System.out.printf("*");
    for(int j=1;j<=n-i;j++)
        System.out.printf("#");
    System.out.printf("\n");
}
```



- break用于退出当前的**整个循环**或者switch，若有多重循环则只退出当前作用域所在的循环。当前循环
- continue用于跳过当前这一次循环，即循环体continue下方的语句不执行，直接进入下一次循环。

```
for(int i=1;i<=10;i++)
{
    Do A
    if(i==3) break;
    Do B
}
```

//A做了3次，B做了2次

```
for(int i=1;i<=10;i++)
{
    Do A
    if(i==3) continue;
    Do B
}
```

//A做了10次，B做了9次，第3次的时候B被跳过了

- while和for**不需要分号来结束语句**，在其**圆括号**后面加分号会直接截断这个语句 (~~啥也没做~~)，意味着花括号中的语句不作为它们的作用域循环执行，而是作为跟他们同级的语句在它们结束之后执行。
- for和while圆括号以及花括号中定义的变量都属于它们的作用域
- for一般用于知道循环次数的循环(counter-controlled)，while一般用于知道循环出口的循环(sentinel-controlled)

数组

<#>

Primitive type 和 reference type

Primitive types

变量中直接存的是需要的数值

byte, short, int, long, char,
boolean, float, double

Reference types

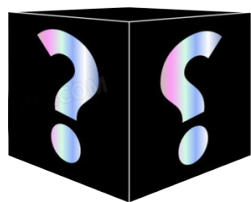
变量中存储的是所需要东西的地址，在通过地址到
对应内存空间寻找所需要的东西

除了primitive type的其他变
量类型

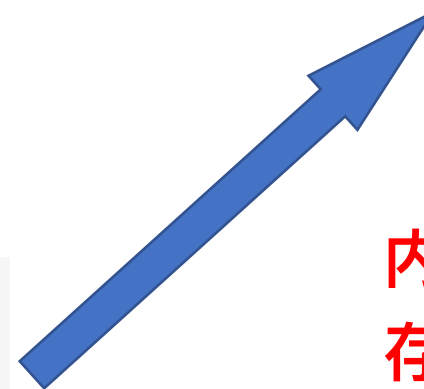
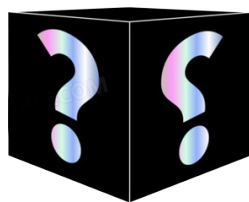
<#>

Primitive type 和 reference type

Primitive Types



Reference Types



内存空间



变量



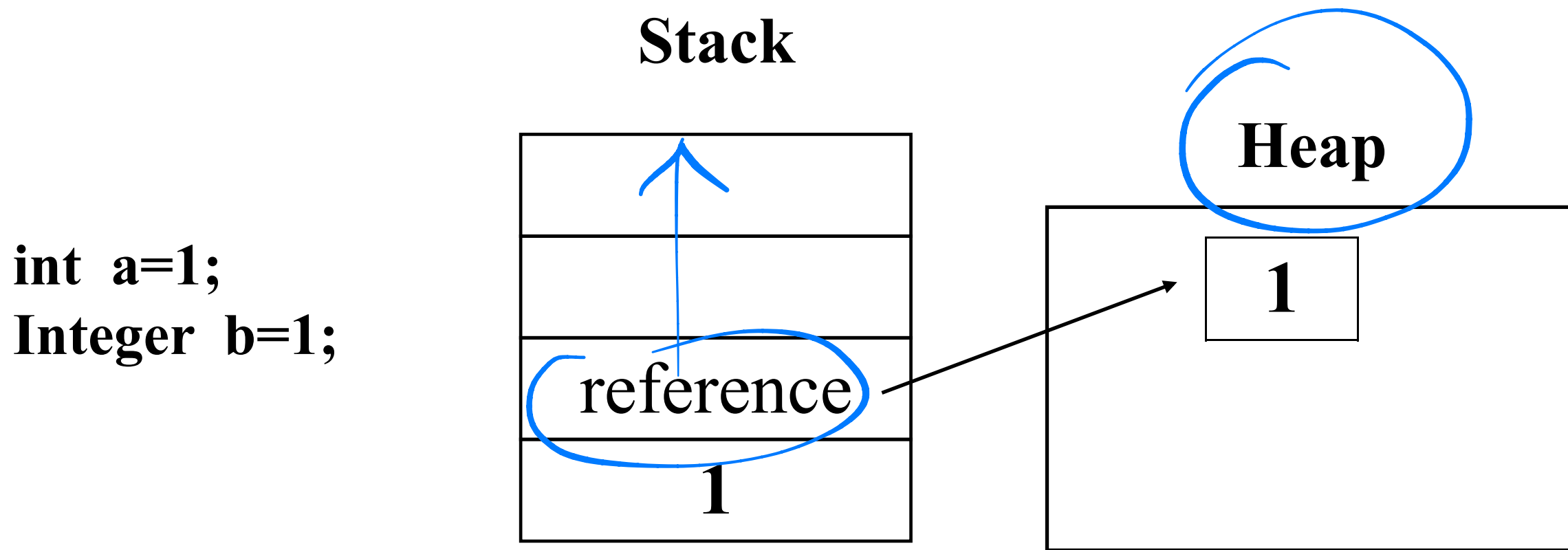
得到的值



地址

<#>

Primitive type 和 reference type



- 变量实际存储的值存在栈(stack)中，引用类型指向的实体存在于堆(Heap)中。

- 数组是一系列相同类型的变量的集合。
- 数组其本身是一个reference type的变量，其地址指向内存中存放数组的位置，数组的元素依次连续存放在其后。但我们说数组一般指的是这一系列元素。

<#>

数组的定义

存储的变量
类型+[]表示
数组

变量名

```
int[] a=new int[10];
```

创建一个大小为10的
int数组，将第0个元素
的地址赋值给变量

▲ 注意:数组从0
开始存储

<#>

数组元素的调用

哪个位置的元素，称为下标

变量名

```
a[i] = 10;
```

取出后就是一个int
类型变量，可以进行
赋值等操作

- 数组元素被调用出来之后就视为该变量类型的一个变量，和普通定义的变量没有区别。
- 当然，调取数组元素的下标时也可以使用变量和运算符构成的表达式来表示
- 每个数组都拥有一个length变量，调用^{属性}a.length可以知道其长度



数组的初始化

- 已知数组内容，直接赋值，赋值后数组长度为给入的数值个数

```
int[] a={1,2,3,4,5,6};
```

- 不知道具体数值，但可以估计变量个数最大不超过多少。利用new生成变量空间，不同变量类型有默认的初始值

```
int[] a=new int[10];
```

<#>

数组复制



```
public class test{

    public static void main (String[] args)
    {
        int[] a={0,1,2,3};
        int[] b=new int[a.length];
        b=a;
        b[1]=2;
        for(int i:a) System.out.printf("%d ",i);
    }
}
```

0 2 2 3

Process finished with exit code 0

```
public class test{

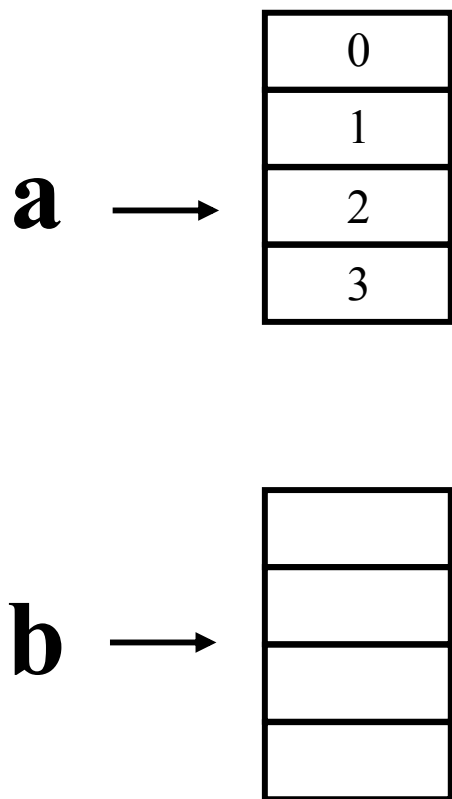
    public static void main (String[] args)
    {
        int[] a={0,1,2,3};
        int[] b=new int[a.length];
        for(int i=0;i<a.length;i++) b[i]=a[i];
        b[1]=2;
        for(int i:a) System.out.printf("%d ",i);
    }
}
```

0 1 2 3

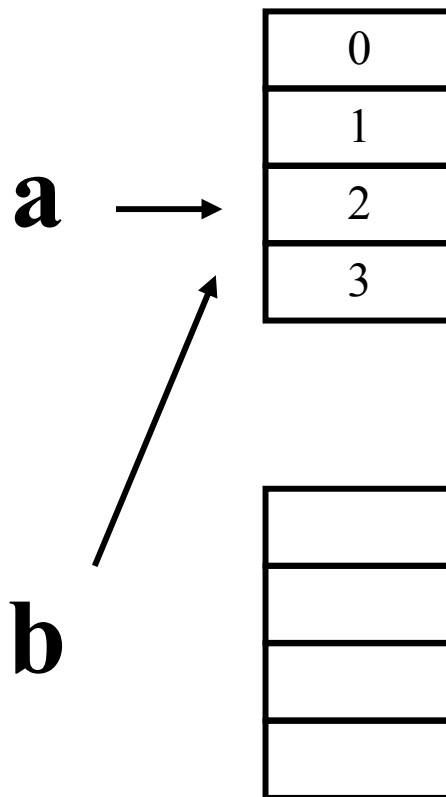
Process finished with exit code 0

<#>

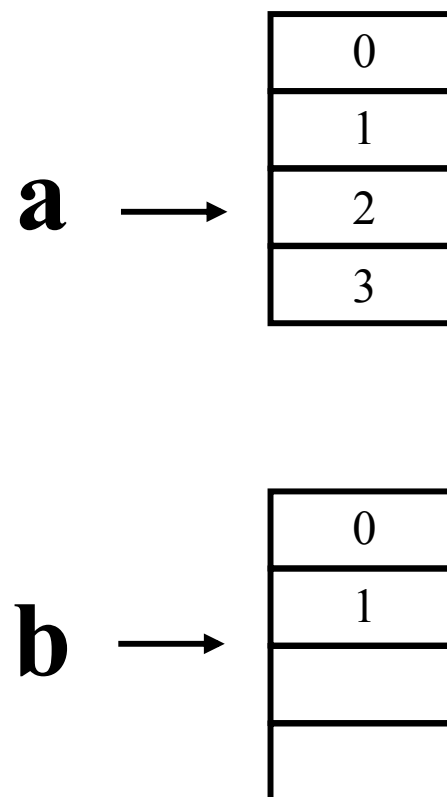
数组复制



```
b=a;
```



```
for(int i=0;i<a.length;i++) b[i]=a[i];
```





如何进行调试

课堂练习



**Thanks for
watching**