* **厦门大学**

**Java上机实验**

**（第二次四个实验）**

**姓名：刘馨欣**

**学号：22920132203820**

**班级：计算机1班**

**系别：计算机**

**学院：信息科学与技术学院**

**实验时间：2015年04月03日**

**报告完成时间：2015年04月10日**

**实验一 最大子方阵**

**（Largest Block）**

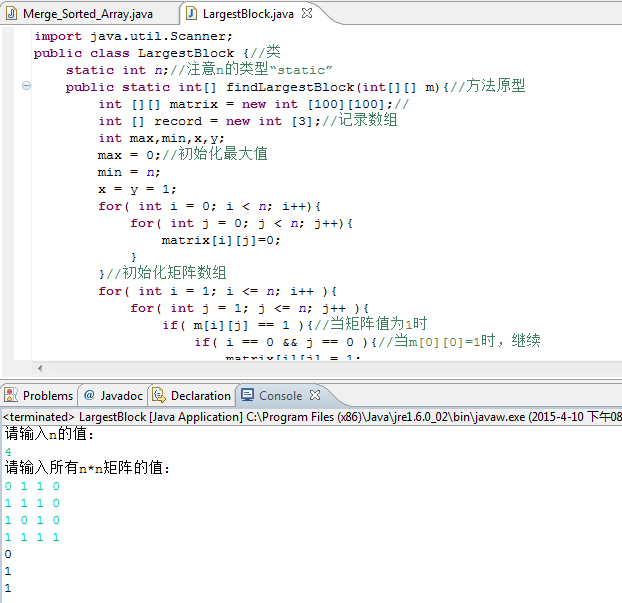
**一、实验题目**

给定一个由0,1组成的n\*n方阵（n在运行时提醒用户输入），判断其中由全1组成的最大子方阵的左上角位置和阶数。例如用户输入n为5，并且输入方阵如下：

要求编写方法实现上述功能，返回值是一个包含**3**个元素的数组，依次表示**行下标**，**列下标**，**阶数。**

方法原型：public static int[] findLargestBlock(int[][] m)

**二、实验结果**



**三、实验设计思路**

本实验设计思路：

1.输入部分，分别输入方阵大小n以及方阵中各元素大小；

2.用record数组进行比较；

3.输出结果。

**四、实验收获**

了解并熟悉了Java中“方法”的使用。

**五、源代码**

**import** java.util.Scanner;

**public** **class** LargestBlock {//类

**static** **int** *n*;//注意n的类型“static”

**public** **static** **int**[] findLargestBlock(**int**[][] m){//方法原型

**int** [][] matrix = **new** **int** [100][100];//

**int** [] record = **new** **int** [3];//记录数组

**int** max,min,x,y;

max = 0;//初始化最大值

min = *n*;

x = y = 1;

**for**( **int** i = 0; i < *n*; i++){

**for**( **int** j = 0; j < *n*; j++){

matrix[i][j]=0;

}

}//初始化矩阵数组

**for**( **int** i = 1; i <= *n*; i++ ){

**for**( **int** j = 1; j <= *n*; j++ ){

**if**( m[i][j] == 1 ){//当矩阵值为1时

**if**( i == 0 && j == 0 ){//当m[0][0]=1时，继续

matrix[i][j] = 1;

**continue**;

}

**if**( i - 1 >= 0 && matrix[i-1][j] < min )

min = matrix[i-1][j];

**if**( j - 1 >= 0 && matrix[i][j-1] < min )

min = matrix[i][j-1];

**if**( i - 1 >= 0 && j - 1 >= 0 && matrix[i-1][j-1] < min )

min = matrix[i-1][j-1];

**if**( min + 1 > max ){

max = min+1;

x = i;

y = j;

}

matrix[i][j] = min+1;

}

**else** matrix[i][j] = 0;

}

}

record[0] = x - max;

record[1] = y - max;

record[2] = max;

**return** record;

}

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

Scanner input = **new** Scanner( System.*in* );

**int** [][] matrix = **new** **int** [100][100];

**int** [] record;

**int** i,j;

System.*out*.print( "请输入n的值:\n" );

*n* = input.nextInt();

**for**( i = 0; i < *n*; i++ ){

**for**( j = 0; j < *n*; j++ ){

matrix[i][j]=0;

}

}

System.*out*.print( "请输入所有n\*n矩阵的值:\n" );

**for** (**int** row = 1; row <= *n*; row++) {

**for** (**int** column = 1; column <= *n*; column++) {

matrix[row][column] = input.nextInt();

}

}

record = *findLargestBlock*( matrix );

**for**( i = 0; i < 3; i++ )

System.*out*.println( record[i] );

}

}

**实验二 二次方程类**

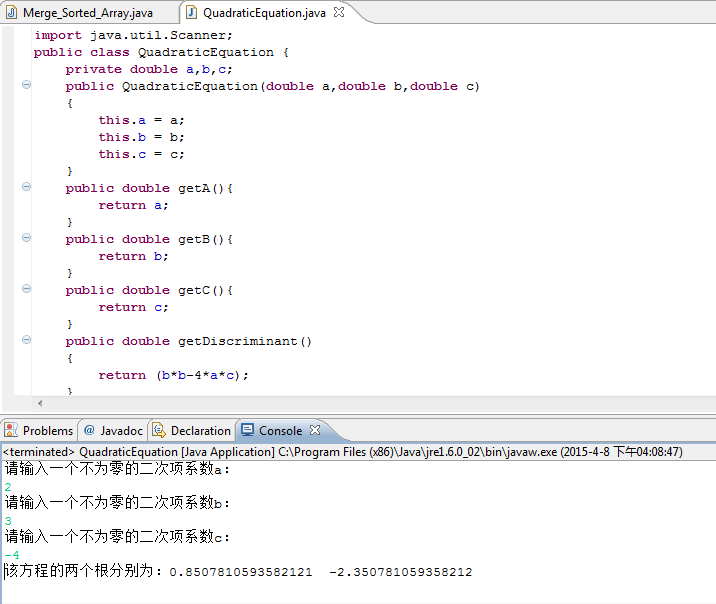
**（Quadratic Equation）**

**一、实验题目**

设计一个二次方程类QuadraticEquation，用于处理形如ax2 + bx + x = 0 (a≠0)的二次方程，成员如下：

* + 私有成员a, b, c用于存储系数；
  + 含三个参数的构造方法，用于传入a, b, c；
  + 三个方法getA(), getB(), getC()，用于传出系数；
  + 一个方法getDiscriminant()用于传出b2 - 4ac的值；
  + 两个方法getRoot1()和getRoot2()用来返回方程的两个根。注意方程可能没有实根，所以返回值定义为String。
  + 请提供一个测试类，测试上述所有方法。

**二、实验结果**



**三、实验设计思路**

本实验在设计思路上十分清晰，在算法设计上难度简单。主要应用了Java中特有的“方法”来进行输入。

**四、实验收获**

学会用Java中特有的“方法”实现输入功能。

**五、源代码**

**import** java.util.Scanner;

**public** **class** QuadraticEquation {

**private** **double** a,b,c;

**public** QuadraticEquation(**double** a,**double** b,**double** c)

{

**this**.a = a;

**this**.b = b;

**this**.c = c;

}

**public** **double** getA(){

**return** a;

}

**public** **double** getB(){

**return** b;

}

**public** **double** getC(){

**return** c;

}

**public** **double** getDiscriminant()

{

**return** (b\*b-4\*a\*c);

}

**public** **double** getRoot1()

{

**if**(**this**.getDiscriminant()>=0)

**return** ((-b+Math.*pow*(**this**.getDiscriminant(), 0.5))/(2\*a));

**else**

**return** 0;

}

**public** **double** getRoot2()

{

**if**(**this**.getDiscriminant()>=0)

**return** ((-b-Math.*pow*(**this**.getDiscriminant(), 0.5))/(2\*a));

**else**

**return** 0;

}

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

Scanner input = **new** Scanner(System.*in*);

**double** x,y,z;

System.*out*.println("请输入一个不为零的二次项系数a： ");

x = input.nextDouble();

System.*out*.println("请输入一个不为零的二次项系数b： ");

y = input.nextDouble();

System.*out*.println("请输入一个不为零的二次项系数c： ");

z = input.nextDouble();

QuadraticEquation qua = **new** QuadraticEquation(x,y,z);

System.*out*.println("该方程的两个根分别为："+qua.getRoot1()+

" "+qua.getRoot2());

}

}

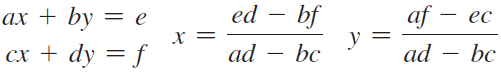
**实验三 2\*2线性方程组**

**（**Linear Equation**）**

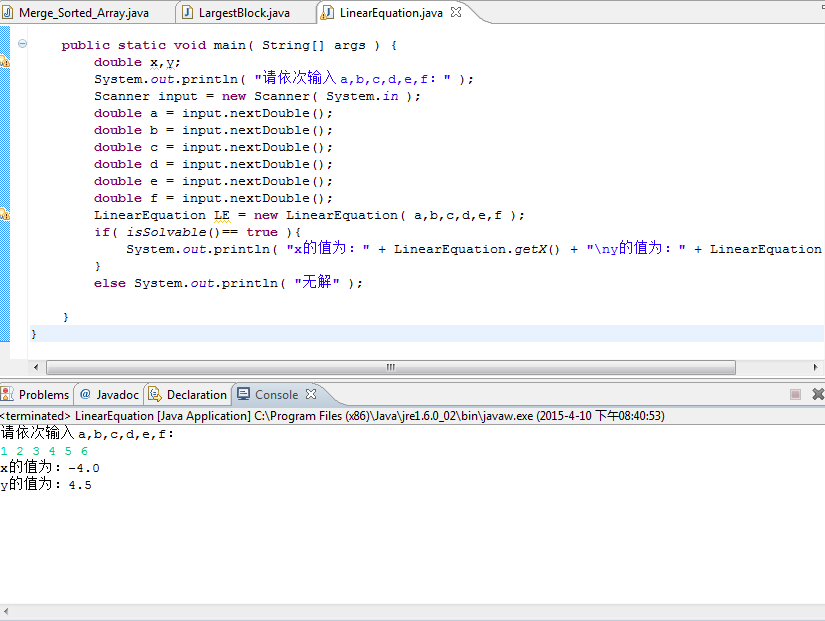
**一、实验题目**

设计一个类LinearEquation用于处理如下的2\*2线性方程组，成员包含：

* + 私有成员a,b,c,d,e,f；
  + 一个6参数构造方法，用于传入a,b,c,d,e,f；
  + 6个getter用于返回a,b,c,d,e,f，例如getA(), getB(),…；
  + 一个方法isSolvable()用于判定方程是否有解，有则返回true，否则false；
  + 方法getX()和getY()返回一组解。
  + 请提供一个测试类，测试上述所有方法。



**二、实验结果**

****

**三、实验设计思路**

与实验二的思路非常类似，主要是巩固对方法的运用。

**四、实验收获**

1.熟悉构造方法；

2.定义时如果不带“static”。。程序就毁了。。虽然依然不太get到为什么。。但是根据Java自带提示将实验二中的“public”一路改为“static”，程序就ok了。。

3.再次遇到boolean类型。。很亲切。。貌似必须用“static”就是因为boolean类型。。

**五、源代码**

**import** java.util.Scanner;

**public** **class** LinearEquation {

**private** **static double** *a*,*b*,*c*,*d*,*e*,*f*;

LinearEquation( **double** a,**double** b,**double** c,**double** d,**double** e,**double** f ){

**this**.*a* = a;

**this**.*b* = b;

**this**.*c* = c;

**this**.*d* = d;

**this**.*e* = e;

**this**.*f* = f;

}

**static** **double** getA(){

**return** *a*;

}

**static** **double** getB(){

**return** *b*;

}

**static** **double** getC(){

**return** *c*;

}

**static** **double** getD(){

**return** *d*;

}

**static** **double** getE(){

**return** *e*;

}

**static** **double** getF(){

**return** *f*;

}

**static** **boolean** isSolvable(){//分母不为0

**if**( *a*\**d*-*b*\**c* == 0 ) **return** **false**;

**else** **return** **true**;

}

**static** **double** getX(){

**double** x=( *e*\**d*-*b*\**f* )/( *a*\**d*-*b*\**c* );

**return** x;

}

**static** **double** getY(){

**double** y = ( *a*\**f*-*e*\**c* )/( *a*\**d*-*b*\**c* );

**return** y;

}

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

**double** x,y;

System.*out*.println( "请依次输入 a,b,c,d,e,f：" );

Scanner input = **new** Scanner( System.*in* );

**double** a = input.nextDouble();

**double** b = input.nextDouble();

**double** c = input.nextDouble();

**double** d = input.nextDouble();

**double** e = input.nextDouble();

**double** f = input.nextDouble();

LinearEquation LE = **new** LinearEquation( a,b,c,d,e,f );

**if**( *isSolvable*()== **true** ){

System.*out*.println( "x的值为：" + LinearEquation.*getX*() + "\ny的值为：" + LinearEquation.*getY*() );

}

**else** System.*out*.println( "无解" );

}

}

**实验四 Location类**

**（Location）**

**一、实验题目**

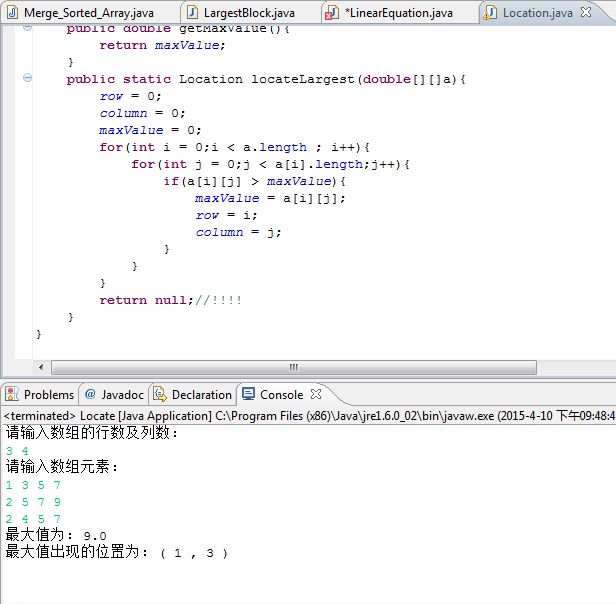
定义一个Location类，用于搜索二维数组的最大元素出现的位置和值。位置用**公有的**整型成员变量row, col表示，最大值用**公有的**浮点型成员变量maxValue表示。一个成员方法用来求解二维数组的最大元素及其位置，原型如下：

public static Location locateLargest(double[][] a)

例如数组为{{1,2,3},{8,9,9,5},{4,3,5,7,8}}，最大元素为9，位置是(1,1)。注意最大值不止一个的时候，只记录第一次出现的位置。

请提供一个测试类，测试上述方法。

**二、实验结果**

****

**三、实验设计思路**

1.输入数组；

2.找到最大元素；

3.定位最大元素位置。

**四、实验收获**

学会应用“类”，复习了查找最大元素的算法。

**五、源代码**

**import** java.util.Scanner;

**class** Locate{

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

System.*out*.println( "请输入数组的行数及列数：" );

Scanner input = **new** Scanner( System.*in* );

**int** row = input.nextInt();

**int** column = input.nextInt();

System.*out*.println( "请输入数组元素：" );

**double**[][]array = **new** **double** [row][column];

**for**( **int** i = 0; i < array.length; i++ ){

**for**(**int** j = 0; j < array[i].length; j++ ){

array[i][j] = input.nextDouble();

}

}//输入数组元素

Location location = **new** Location();

Location.*locateLargest*( array );

System.*out*.println( "最大值为： " + location.getMaxValue() + "\n最大值出现的位置为： ( " + location.getRow() + " , " + location.getColumn() + " ) " );

}

}

**class** Location{

**private** **static** **int** *row*;

**private** **static** **int** *column*;

**private** **static** **double** *maxValue*;

**public** **int** getRow(){

**return** *row*;

}

**public** **void** setRow(**int** row){

**this**.*row* = row;

}

**public** **int** getColumn(){

**return** *column*;

}

**public** **void** setCloumn(**int** column){

**this**.*column* = column;

}

**public** **double** getMaxValue(){

**return** *maxValue*;

}

**public** **static** Location locateLargest(**double**[][]a){

*row* = 0;

*column* = 0;

*maxValue* = 0;

**for**(**int** i = 0;i < a.length ; i++){

**for**(**int** j = 0;j < a[i].length;j++){

**if**(a[i][j] > *maxValue*){

*maxValue* = a[i][j];

*row* = i;

*column* = j;

}

}

}

**return** **null**;//!!!!

}

}