Mathematical Algorithms

Arrays and Matrices

When we declare an array, we must first state a type of values to be stored in array, followed by a pair of brackets (marking, it is a field) and the array name.

When initializing manually, type an equal sign followed by the keyword **new** and the name of type stored in the array.

Then, in brackets, follows signed integer value that defines how the field is large.

Example

```
//array of eight integers
int[] array = new int[8];

//array of four strings
String[] stringArray = new String[4];
```

Accessing members of Array

To access the values stored in the array we use square brackets for the inscribed name of a field variable. In these brackets we index (zero-based) of the selected value.

Example

```
//array of eight integers
int[] array = new int[8];
array[0] = 5; //first value is 5
array[1] = 3; //second value is 3
array[2] = array[1]; //third value is same as array[1]

array[1] = 4; //array[1] == 4, array[2] == 3

//array of four strings = new String[4];
stringArray[0] = "Alice";
```

```
stringArray[1] = "Ben";
stringArray[2] = "Draig";
stringArray[3] = "David";
System.out.println(stringArray[0]); //Alice
```

Simplified initialization

Such initialization is a very lengthy and clumsy. Therefore, we have an option to join the declaration with initialization to create a field in an alternative manner in which the right side bears the inscription in curly brackets comma separated values to which field should be initialized to the individual indices.

Example

```
String[] stringArray2 = {"Alice", "Ben",
"Craig", "David"}; //pole ctyr retezcu
```

Example

The quadratic formula can have 0-2 solutions in the field of real numbers depending on if discriminant is negative, zero or positive.

```
/**
 * Solves quadratic fomula in the form
 * ax^2 + bx + c = 0
 * @param a
 * @param b
```

```
 * @param c
 * @return array of real roots
 */
public static double[]
solveQuadraticEquation(double a, double b,
double c) {
    double d = b*b - 4*a*c; //discriminant
    if (d < 0) {
        double[] result = new double[0];
        return result;
    } else if (d == 0) {
        double[] result = {-b/2*a};
        return result;
    } else {
```

```
        double[] result = {(-b +
Math.sqrt(d))/(2*a), (-b -
Math.sqrt(d))/(2*a)};
        return result;
    }
}
```

Array and *for* loop

Structure *for-each* loop is similar to the the *for* cycle, with the difference that in parentheses find firstly type of variables stored in the collection through which iterate, followed by the variable name, under which they will be made available at each

iteration stored objects, colon and variable name indexing the collection.

```
/**
 * Prints all values stored in the array
 * @param array
 */
public static void print(String[] array){
    for (String s : array) {
        System.out.println(s);
    }
}
```

Naturally, classic *for* loop can be used as well:

```
/**
 * Prints all numbers stored in an array
 * @param array
 */
public static void print(int[] array) {
    for (int i = 0; i < array.length; i++)
{
        System.out.println(array[i]);
    }
}
```

Matrices

Matrix is a multidimensional array.

Declaring multidimensional arrays is similar to that of a one-dimensional array, the only difference is the number of brackets (one pair of parentheses for one dimension). On the right side of the assignment **you must always specify the size of the first dimension** (the actual array of fields). The size of other dimensions do not have to fill in, but then Java cannot initialize these dimensions (this must be done manually).

The omission of other dimensions may be useful if we want to create some way irregular array - for example in the shape of a triangle.

For multidimensional arrays apply the following guidance: two-dimensional array is a common, three-dimensional is unusual and more than three-dimensional error is usually the application design (only a few people can imagine such data).

Example

```
//array 3 rows, 4 columns
int[][] array2d = new int[3][4];
```

```
int[][] array2d2 = new int[3][]; //second
dimension is not shown
for(int i = 0; i < array2d2.length; i++){
//triangle matrix
    array2d2[i] = new int[i + 1];
}
```

Example

```
/**
 * Prints out a matrix (2-dimensional
array)
 * @param array matrix
 */
```

```
public static void print(int[][] array) {
    for (int i = 0; i < array.length; i++)
{ //1st dimension
        for (int j = 0; j <
array[i].length; j++) { //2nd dimension
            System.out.print(array[i][j] +
" "); //no CR/LF
        }
        System.out.println(""); //new line
    }
}
```

Initialization can be simplified as well.

```
int[][] setting = {
```

```
    {0, 0, 4, 0, 3, 6, 9, 2, 7},
    {1, 0, 0, 0, 0, 5, 0, 0, 0},
    {0, 0, 0, 2, 0, 0, 0, 0, 4},
    {0, 0, 5, 0, 0, 0, 0, 6, 0},
    {6, 4, 0, 0, 0, 0, 0, 8, 5},
    {0, 7, 0, 0, 0, 0, 2, 0, 0},
    {5, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 0, 7, 0, 0, 0, 0, 2},
    {4, 3, 7, 9, 2, 0, 5, 0, 0}
};
```

Multiplication of Matrices

Matrix multiplication actually is an extension of the usual multiplication of two numbers on the case of two matrices.

If A is a matrix of dimensions $m \times n$ and B matrix is $n \times p$, then the product A * B is the matrix of dimensions $m \times p$, defined as:

$$(A \cdot B)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}.$$

Example

Suppose matrices

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

The product is

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} (1 \cdot 1 + 2 \cdot 3 + 3 \cdot 5) & (1 \cdot 2 + 2 \cdot 4 + 3 \cdot 6) \\ (4 \cdot 1 + 5 \cdot 3 + 6 \cdot 5) & (4 \cdot 2 + 5 \cdot 4 + 6 \cdot 6) \end{pmatrix} = \begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$$

```
for(int i = 0; i < matA.length; i++) {
    for(int j = 0; j < matB[0].length; j++)
{
        int sou = 0;
        for(int s = 0; s<matB.length; s++){
            sou += (matA[i][s]*matB[s][j]);
            }
        vynasmat[i][j] = sou;
```

```
            }
        }
```

When we declare an array, we must first state a type of values to be stored in array, followed by a pair of brackets (marking, it is a field) and the array name. When initializing manually, type an equal sign followed by the keyword **new** and the name of type stored in the array. Then, in brackets, follows signed integer value that defines how the field is large.

GCD of two numbers is the largest positive integer that divides both the numbers fully i.e. without any remainder. There are multiple methods to find GCD , GDF or HCF of two numbers but Euclid's algorithm.

Euclid's algorithm is an efficient way to find GCD of two numbers and its pretty easy to implement using recursion in Java program. It is one of the oldest known nontrivial algorithms and gradually, a number of modifications thereof, for example related tasks.

Example

Consider two numbers, 133 and 15, their highest common divisor (GCD) can be found as follows:

133 divide by 15, left 13:

133 * 8 = 15 + 13

Now we have to realize these things. Both left and right sides of the equation must be divisible by gcd (gcd because we are looking for 133 and 15 and both sides are equal to 133).

Another finding is that the multiple of 15 is insignificant, because gcd itself is factor of number 15, or multiple (here 8) we may skip.

The last and most important finding is that if gcd divides into 15 and the right side is divisible by gcd then the residue (in this case 13) must be divisible by gcd.

When we combine all these rules, follows that dividing with a remainder can be repeated for the reduced numbers. Since the

left side must always be smaller and can not drop to 0, then the algorithm is finite.

Another approach is therefore analogous to the first step, only the lower numbers. Gradually, we get:

15 = 1 * 13 + 2

13 = 6 * 2 + 1

2 = 2 * 1 + 0

Now, when we obtained the result 0, we see that the highest number that divides both the left and right side, it is the number 1 (because in the previous step, we found that the wanted number must divide both numbers 2 and 1).

Another example

Numbers 140 and 15:

140 = 9*15 + 5

15 = 3*5 + 0

GCD of 140 and 15 is 5.

```
public class GCDExample {

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

```
        //Enter two number whose GCD needs
to be calculated.
        Scanner scanner = new
Scanner(System.in);
        System.out.println("Please enter
first number to find GCD");
        int number1 = scanner.nextInt();
        System.out.println("Please enter
second number to find GCD");
        int number2 = scanner.nextInt();

        System.out.println("GCD of two
numbers " + number1 +" and "
                                + number2 +" is
:" + indGCD(number1,number2));
```

```
    }

    /*
     * Java method to find GCD of two
number using Euclid's method
     * @return GDC of two numbers in Java
     */
    private static int findGCD(int number1,
int number2) {
        //base case
        if(number2 == 0){
            return number1;
        }
        return findGCD(number2,
number1%number2);
```

```
    }
}
```

Result

```
Please enter first number to find GCD
54
Please enter second number to find GCD
24
GCD of two numbers 54 and 24 is :6
```