3. Arrays

Arrays are a fundamental data structure in each programming language. An array is a data structure that stores a collection of values of the same type. You access each individual value through an integer *index* specifying the position of the value in the array. Compared to C, C++ or FORTRAN, arrays in Java are much easier to use:

- Java arrays know how large they are. You do not have to pass the length of the array to a function as an extra argument.
- A bounds checking mechanism gives an error in the case of an out of bounds access to the array. This greatly facilitates software development because bugs are much easier to find this way.

↳ Nizovi pohranjuju skup podataka istog tipa
↳ Svakom elementu niza se može pristupiti kroz integer index koji specificira poziciju vrijednosti u nizu
↳ Java nizovi znaju koliki su (ne moramo ni definisati)
↳ "Bounds checking mechanism" daje grešku u slučaju pokušaja pristupa nizu van granica

You declare an array variable by specifying the type of the values in the array followed by `[]` and the name of the variable. For example, here is the declaration of an array of boolean values:

```
boolean[] b;
```

However, this statement only declares the variable b to be an array of booleans. It does not yet initialize it. Because arrays are some sort of objects in Java (they share a lot with objects, but differ in some aspects), an array must also be created via `new`. The number of elements of the array is given in brackets. For example, the statement

```
double[] x = new double[20];
```

[] umjesto () kao kod normalnih objekata

creates an array of 20 doubles and stores it in the variable x. During array creation, the array elements are initialized to their default values (which is zero for the numerical primitive types). Note that the difference to the creation of "normal" objects is that brackets `[]` are used instead of the parenthesis `()`. Access to an element of an array looks like

```
x[0] = 44.2;
double y = x[19];
```

where the first statement stores a value in the array and the second reads a value from the array. Note that array indexing is zero based in Java (other than in FORTRAN) and thus the statement

```
x[20] = 2.0;
```

results in an error message and program termination (remember that the array bounds are checked during runtime).

↳ DEFINIŠU SE KAO

TIP_VRIJEDNOSTI [] IME_VARIJABLE;

↳ No ovaj način se samo neka varyabla deklarište nekom nizu ⇒ ne inicyalizira se

↳ Kreiraju se pomoću new

veličina u zagradi

TIP_VRIJEDNOSTI [] IME_VARIJABLE = new TIP_VRIJEDNOSTI [size];

↳ Pri izradi niza sa new, elementi niza se inicyaliziraju na default vrijednosti (0 za numeričke primitivne tipove)

↳ Da bi pristupili elementu unosimo njegov indeks u zagradi y = x [7]

↳ Prvi indeks je 0!

↳ Za pronalazak broja elemenata koristimo

x.getlength

To find the number of elements in an array named x, use x.length. A typical example is to process each element of the array in a for–loop:

```
for(int i = 0; i < x.length; i++) {
  x[i] = 1.0/(i+1);
}
```

↳ Za obradu niza koristimo petlje, npr. for petlju

Arrays can not only be created for the Java primitive types as above, but also for object type variables (remember that each class defines a type). Thus, the statement

```
String[] days = new String[7];
```

defines an array of strings. The values of object type arrays are initialized to null which indicates that they do not refer to an object yet. Access to the array elements is similar as for primitive types:

```
days[0] = "Monday";
```

To be complete, it should also be mentioned that two-dimensional arrays can be created in Java. The statement

```
int[][] a = new int[30][10];
```

declares a matrix of 30×10 integers. To access the elements, two pairs of brackets must be used:

```
a[3][8] = 444;
```

Assignment:

Assignments. In order to become comfortable with arrays, create a class `ArrayTest`, add a `main` method and take the following points as a guideline:

1. Enter the statements above.
2. Print the values of the array `x`.
3. Compute the sum of the entries of `x` and print it. (Answer: 3.597...)
4. Make the array `days` complete and print it.

The Parameter List of the Main-Method. At the beginning of the course you might have wondered about the `String[] args` in the argument list of the `main` method. Now you know that it is an array of strings! The argument can be used to pass some information to the program at startup.

Solution:

```
package chapter1;

public class ArrayTest {

    public static void main (String [] args) {
        boolean [] b;
        double [] n = new double [20];
        n[0]=44.2;
        double y = n [19];

        //For loop for setting the values of the array
        for (int i=0; i< n.length; i++) {
            n [i] = 1.0/(i+1);
        }

        //for loop for printing the array
        for (int i=0; i < n.length; i++) {

            System.out.println("The elementh of an array with the index " + i + " is x[" + i + "] = " + n[i] + "\n");
        }

        //Calculating the sum of the elements of the array
        double sum = 0;
        for (int i=0; i< n.length; i++) {
            sum= sum + n [i];
        }
        System.out.println("The sum of elements of the array is " + sum + "\n");

        //Defining an array whose elements are the days in the week
        String [] days = new String [7];
        days [0]= "Monday";
        days [1]= "Tuesday";
        days [2]= "Wednesday";
        days [3]= "Thursday";
        days [4]= "Friday";
        days [5]= "Saturday";
        days [6]= "Sunday";

        System.out.println("Days in the week are :");
        //Printing that array
        for (int i=0; i<days.length; i++) {
            System.out.println(days[i]);
        }
    }

}
```

Exercises

3.2.1 Vector

Complete the Java implementation of the `Vector` class developed in the lecture using the following information:

- Define an attribute of type `double[]` to store the actual values of the vector.
- Define two constructors: A constructor with a parameter n (which creates an array of n double values initialized to the number 0) and a second constructor with a parameter of type `double[]`.
- Define a `get` and `set` method to get and set a value at a given index, respectively.
- Define a `getN` method to return the dimension or length.
- Define a method `oneNorm` to calculate the Manhattan norm

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|.$$

- Define a method `maxNorm` to calculate the maximum norm

$$\|x\|_\infty := \max(|x_1|, ..., |x_n|).$$

- Define a method `add` to add another vector **y** scaled with a value α to this vector object **x**,

$$x_i = x_i + \alpha y_i, \quad i = 1, ..., n.$$

- Define amethod `multiply` to multiply this vector object **x** by a scalar α,

$$x_i = \alpha x_i, \quad i = 1, ..., n.$$

Specify for each method some useful test cases (test programs, test units) to verify your implemention.

Vector class - develop a class which allows us to create and work with objects representing vectors

$$\underline{X} = (X_1, X_2, \ldots, X_n)^T$$

Common operators

- Construct - create a vector using either the size or by passing the vector elements
- Acces - get vector size, get- and set vector elements
- Output - print a vector

Mathematical operations

- Multiply a vector by a scalar:

  $\mathbf{y} = \alpha\mathbf{x}$ where $y_i = \alpha x_i$

- Add two vectors (second operand scaled by a factor):

  $\mathbf{z} = \mathbf{x} + \alpha\mathbf{y}$ where $z_i = x_i + \alpha y_i$

- Compute scalar product:

  $(\mathbf{x}, \mathbf{y}) = \sum x_i y_i$

- Compute vector norms:

$\|\mathbf{x}\|_2$	$=$	$\sqrt{(\mathbf{x}, \mathbf{x})}$	two norm or Euclidean norm
$\|\mathbf{x}\|_1$	$=$	$\sum \|x_i\|$	one norm
$\|\mathbf{x}\|_\infty$	$=$	$\max(x_1, x_2, \ldots, x_n)$	maximum norm or infinity norm

Vector (class name)

- elements: double[] (elementi vektora)

+ Vector (n: int)
+ Vector (x: double)
} Dva tipa konstruktora

+ getN(): int → metoda za odredivanje

+ fill (v: double): void
+ set (idx: int, v: double): void
+ get (idx: int): int
+ print (l: String): void
+ multiply (alpha: double): Vector
+ add (alpha: dduble, y: Vector): Vector
+ scalarProduct (y: Vector): double
+ twoNorm (): double
+ oneNorm (): double
+ maxNorm (): double

1) Define an attribute of type double[] to store the actual values of a vector

```
public class Vector {
    private double [] elements;
    ⋮
```

2)

- Define two constructors: A constructor with a parameter n (which creates an array of n double values initialized to the number 0) and a second constructor with a parameter of type double[].

```
public class Vector {
    private double [] elements;
    public Vector (int n) {
        this.elements = new double [n];
    }
    public Vector (double ... x) {
        this.elements = x;}
```

dva konst

elementi su default niz (elementi su 0) od unesenog broja elemenata

key!!!

→ Možemo imati 2 konstruktora ako im je ulazni parametar drugačiji!!!

3) Define get and set methods to get/set values at a given index

```
public double get (int idx) {
    return this.elements[idx]; }

void set(int idx, double v) {
    this.elements[idx] = v; }
```

4)

- Define a `getN` method to return the dimension or length.

```
public int getN() {
    return this.elements.length; }
```

5)

- Define a method `oneNorm` to calculate the Manhattan norm

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|.$$

```
public double oneNorm {
    double sum = 0.;
    for (int i=0; i<this.getN(); i++) {
        sum = sum + Math.abs (x[i]);
    }
return sum; }
```

6)

- Define a method `maxNorm` to calculate the maximum norm

$$\|x\|_\infty := \max(|x_1|, \ldots, |x_n|).$$

```
public double maxNorm() {
    double max = Math.abs(this.elements[0]);
    for (int i=0; i< this.getN(); i++){
        if (Math.abs(this.elements[i]) > max) {
            max = Math.abs (this.elements[i]);
        }
    return max; }
```

7) • Define a method `add` to add another vector **y** scaled with a value α to this vector object **x**,

$$x_i = x_i + \alpha y_i, \quad i = 1, \ldots, n.$$

```
public Vector add (double alpha, Vector v) {
    int n = this.getN();
    double [] x = new double [n];
    for (int i=0; i<n; i++) {
        x[i] = this.elements[i] + alpha * v.elements[i];
    }
Vector res = new Vector (x);
return res; }
```

8) • Define amethod `multiply` to multiply this vector object **x** by a scalar α,

$$x_i = \alpha x_i, \quad i = 1, \ldots, n.$$

```
public Vector multiply (double alpha) {
    int n = this.getN();
    double[] x = new double [n];
    for (int i=0; i<n; i++) {
        x[i] = alpha * this.elements[i];
    }
Vector v = new Vector (x);
return x; }
```

ubacat demo metode definisane u prošlavaju

9) Fill metoda - postavlja sve elemente na poeseen vrednost

```
public void Fill (double v) {
  for (int i=0; i< this.getN(); i++) {
    this.elements[i] = v   }}
```

10) Scalar product

```
public double scalar (Vector y) {
   double S=0
     For (int i=0; i< this.getN(); i++){
      S = S + this.get(i) +y.get(i) ;
        }
     return s; }
```

11) Two norm

```
public double twoNorm() {
   return Math.sqrt( this.scalarProduct(this));
      }
```

12) Print

```
Void print (String s) {
  System.out.println(s + " = (");
    for (int i=0; i<this.getN(); i++) {
        if (i< (this.getN() && i!=(this.getN()-2)) {
      System.out.print(this.elements[i]+ ",");
  }else if (i< this.getN()-1 && i== (this.getN() -2)) {
        System.out.print (this.elements[i]);
  } else {
    System.out.print (", "+this.elements[i]+")^T);}
  }}
```

Matrix class

Complete the Java implementation of the `Matrix` class developed in the lecture using the following information:

- Define an attribute of type `double[][]` to store the actual values of the matrix.
- Define three constructors: A constructor with two parameters n and m (which creates a new array of m rows and n columns), a second constructor with a parameter of type `double[][]` and a third constructor with a parameter of type `Matrix` (which creates a copy of the parameter).
- Define the methods `get` and `set` to get and set a value at a given position in the matrix, respectively.
- Define a method `getM` to return the number of rows.
- Define a method `getN` to return the number of columns.
- Define a method `setRow` to set the matrix row.
- Define a method `transpose` to calculate and return the transpose of the matrix.
- Define a method `maxNorm` to calculate and return the maximum norm

$$\|a\|_\infty := \max(|a_{11}|, |a_{12}|, ..., |a_{mn}|).$$

- Define a method `add` to add another matrix **B** scaled with a value α to this matrix object **A**,

$$a_{ij} = a_{ij} + \alpha b_{ij}, \quad i = 1, ...m, \; j = 1, ...n.$$

- Define a method `multiply` to multiply the current matrix **A** with a vector **x** to get a new vector **y**, i.e. calculate

$$\mathbf{y} = \mathbf{Ax}.$$

- Define a second `multiply` method to multiply the current matrix **A** with another matrix **B** to get a new matrix **C**, i.e. calculate

$$\mathbf{C} = \mathbf{AB}.$$

Specify for each method some useful test cases (test programs, test units) to verify your implemention.

From the lecture:

Objective

Develop a class which allows us to create and to work with objects representing matrices

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{pmatrix}$$

Common operations

- Construct: Create a matrix using either the size or by passing the matrix elements
- Access: get matrix size, get- and set matrix elements
- Output: print a matrix

- Transpose a matrix:

$$\mathbf{B} = \mathbf{A}^T \quad \text{where} \quad b_{ij} = A_{ji}$$

- Compute the maximum norm (or infinity norm):

$$\|\mathbf{A}\|_\infty = \max(|a_{ij}|)$$

- Add two matrices:

$$\mathbf{C} = \mathbf{A} + \alpha\mathbf{B} \quad \text{where} \quad c_{ij} = a_{ij} + \alpha b_{ij}$$

- Multiply a $m \times n$ matrix and a vector of size n:

$$\mathbf{y} = \alpha\mathbf{A}\mathbf{x} \quad \text{where} \quad y_i = \alpha \sum_{j=1}^{n} A_{ij} x_j, \quad i = 1, \dots, n$$

- Multiply a $m \times p$ matrix and a $p \times n$ matrix:

$$\mathbf{C} = \alpha\mathbf{A}\mathbf{B} \quad \text{where} \quad c_{ij} = \alpha \sum_{k=1}^{p} A_{ik} C_{kj}, \; i = 1, \dots, m, \; j = 1, \dots, n$$

UML class diagram:

Matrix

- elements [][]

+ Matrix (m:int, n:int)
+ Matrix (m:int, n:int, vs:double ...)
+ getM():int
+ getN():int
+ set(i:int, j:int, wert:double):void
+ setRow(i:int, vs:double ...):void
+ get(i:int, j:int):double
+ print(s:String):void
+ transpose():Matrix
+ MaxNorm():double
+ add(alpha:double, b:Matrix):Matrix
+ multiply(alpha:double, x:Vector):Vector
+ multiply(alpha:double, b:Matrix):Matrix

Nassi-Shneidermanov diagramen za multiply metode

multiply(alpha: double, x: Vector): Vector
r = new Vector(getM())
i = 0; i < getM(); i = i + 1
s = 0
j = 0; j < getN(); j = j + 1
s = s + get(i, j) * x.get(j)
r.set(i, alpha * s)
return r

multiply(alpha: double b: Matrix): Matrix
r = new Matrix(getM(), b.getN())
i = 0; i < getM(); i = i + 1
j = 0; j < b.getN(); j = j + 1
s = 0
k = 0; k < getN(); k = k + 1
s = s + get(i, k) * b.get(k, j)
r(i, j, alpha * s)
return r

1) • Define an attribute of type `double[][]` to store the actual values of the matrix.

```
public class Matrix {
    private double [][] elements;
}
```

2) • Define three constructors: A constructor with two parameters n and m (which creates a new array of m rows and n columns), a second constructor with a parameter of type `double[][]` and a third constructor with a parameter of type `Matrix` (which creates a copy of the parameter).

```
public Matrix (int m, int n) {
    this.elements = new double [m][n]; }

public Matrix (double [][] x) {
    int n = x.length;
    int m = x[0].length;
    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
            this.elements [i][j] = x[i][j];
        }}
    }

public Matrix copyMatrix (Matrix M) {
    Matrix res = new Matrix (M.elements);
    return res; }
```

3) • Define the methods `get` and `set` to get and set a value at a given position in the matrix, respectively.

```
public double get (int n, int m) {
    return this.elements [n][m]; }

void set (int n, int m, double v) {
    this.elements [n][m] = v; }
```

4) • Define the methods `get` and `set` to get and set a value at a given position in the matrix, respectively.

```
public int getM() {
    return this.elements.length; }

public int getN() {
    return this.elements[0].length; }
```

5) • Define a method `setRow` to set the matrix row.

```
void setRow (int i; double vs) {
    int n = this.elements[i].length;
        for (int j=0; j<n; j++) {
            this.elements[i][j] = vs; } }
```

6) • Define a method `transpose` to calculate and return the transpose of the matrix.

```
public Matrix transpose () {
    int n = this.elements.length;
    int m = this.elements [0].length;
     double [][] Mte = new double [m][n];
      for (int i=0; i<m; i++) {
         for (int j=0; j<n; j++) {
         Mte = this.elements[j][i];
             }}
    Matrix Mt = new Matrix (Mte);
    return Mt; }
```

7) We will define, print method

```
void print (String s) {
  System.out.println ("Defined matrix is "+s+":");
  For (int i=0; i<this.elements.length; i++) {
    for(int j=0; j<this.elements[0].length; j++) {
  if (j==0){System.out.print("| "+this.elements[i][j]+", ");
  } else if (j==this.elements[0].length-1) {
  System.out.print (this.elements[i][j]+ "    |"+"\n");
  } else {
  System.out.print(this.elements[i][j] + ",    " );}
}}
```

8)

- Define a method `maxNorm` to calculate and return the maximum norm

$$\|a\|_\infty := \max(|a_{11}|, |a_{12}|, ..., |a_{mn}|).$$

```
public double maxNorm () {
    double max = this.elements[0][0];
    For (int i=0; i<this.elements.length; i++){
      for(int j=0; j<this.elements[0].length; j++){
        IF (Math.abs(this.elements[i][j])>max) {
        max = Math.abs(this.elements[i][j]);
        }}} return max;
      }
```

9)

- Define a method `add` to add another matrix **B** scaled with a value α to this matrix object **A**,

$$a_{ij} = a_{ij} + \alpha b_{ij}, \quad i = 1, ...m, \; j = 1, ...n.$$

```
public Matrix add (double alpha, Matrix M) {
    int n = this.elements.length;
    int m = this.elements[0].length;
    Matrix resM = new Matrix (n,m);
    for (int i=0; i < n; i++) {
        for (int j=0; j<m; j++) {
            resM.elements[i][j] = this.elements[i][j] + M.elements[i][j]*alpha;
        }
    }
    return resM;
}
```

10)

- Define a method `multiply` to multiply the current matrix **A** with a vector **x** to get a new vector **y**, i.e. calculate

$$\mathbf{y} = \mathbf{A}\mathbf{x}.$$

```
public Vector multiply (Vector v) {
    int n = this.elements.length;
    int m = this.elements[0].length;
    Vector resM = new Vector (n);
    for (int i=0; i < n; i++) {
    double sum=0;
        for (int j=0; j<m; j++) {
        sum = sum + this.elements[i][j] * v.get(j);
        }
    resM.set(i,sum);
    }
    return resM;
}
```

11)

- Define a second `multiply` method to multiply the current matrix **A** with another matrix **B** to get a new matrix **C**, i.e. calculate

$$\mathbf{C} = \mathbf{AB}.$$

```
public Matrix multiply (Matrix M) {
    int n = this.elements.length;
    int m = this.elements[0].length;
    Matrix resM = new Matrix (n,m);
    for (int k=0; k<n; k++) {

    for (int i=0; i < n; i++) {
        double sum=0;
            for (int j=0; j<m; j++) {
            sum = sum + this.elements[k][j] * M.elements[j][i];
            }
        resM.elements[k][i]=sum;
    }}
        return resM;
}
```