Sorting Arrays with bubble sort and Introduction to Multidimensional arrays

Objectives

- Recap previous session
- Sorting Arrays
- Introduction
- Class exercises
- Homework exercises
- Guidelines

Recap previous session

- What is an Array?
- Why and when do we need an array?
- An array of int, is a primitive or an object? Why?
- How can we find how many elements can we store inside an array?

Sorting arrays

- Often, we have to get into sorting the values in an array either ascending or descending.
- There are multiple solutions with regard to sorting an array. The most common and the one that we will talk about today is `BubbleSort`.
- The idea behind `BubbleSort` is that we swap adjacent elements if they are in the wrong order
- Consider the example below in which we try to sort the following array in ascending order:
    - The array is: `[5 1 4 2 8]`
    - Algorithm:

      `First Pass`:

      (`5 1` 4 2 8) –> (`1 5` 4 2 8) -> Here, algorithm compares the first two elements, and swaps since 5 > 1.

      (1 `5 4` 2 8) –> (1 `4 5` 2 8) -> Swap since 5 > 4

      (1 4 `5 2` 8) –> (1 4 `2 5` 8) -> Swap since 5 > 2

      (1 4 2 `5 8`) –> (1 4 2 `5 8`) -> Now, since these elements are already in order (8 > 5), algorithm does not swap them.

      `Second Pass`:

      (`1 4` 2 5 8) -> (`1 4` 5 2 8) -> The elements are in order, algorithm does not swap them

(1 4 2 5 8) -> (1 2 4 5 8) -> Swap since 4 > 2

Note: Our array is already sorted but this is the disadvantage of the Bubbleort algorithm. It will continue to check the elements in the following order:

(1 2 4 5 8) -> (1 2 4 5 8) -> The elements are in order, algorithm does not swap them

(1 2 4 5 8) -> (1 2 4 5 8) -> The elements are in order, algorithm does not swap them

(1 2 4 5 8) -> (1 2 4 5 8) -> The elements are in order, algorithm does not swap them

(1 2 4 5 8) -> (1 2 4 5 8) -> The elements are in order, algorithm does not swap them

- Now, below you have the JAVA program which sorts the array above:

```
public class Application {

public static void main(String[] args) {
  int arr[] = new int[] {5,1,4,2,8};

  for(int i = 0; i < arr.length; i++) {
    for(int j = 0; j < arr.length-1;j++) {
      if(arr[j] > arr[j+1]) {
        int temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
      }
    }
  }

  //Now display the array which should be sorted by now
  for(int i = 0; i < arr.length; i++) {
    System.out.print(arr[i] + " ");
  }
 }
}
```

- Now we should also explain what happened in the previous snippet:
  - We start by creating an array with values automatically populated by us (in the homework exercises you will have to read the values from the standard input)
  - Next we move to the logic from the algorithm which compares each adjacent elements to see the relation between them. We need two for's because we compare each element with each other element. You can see it like this:
    - For i = 0:
      - compare with first element
      - compare with second element
      - compare with third element
      - compare with fourth element

        - compare with fifth element
    - For i = 1
        - repeat previous steps
    - etc.

- This guarantee us that in the end, we have compared all elements between them. Note that the second for goes until lentgh-1 because we compare the current element with the one on the next position (j + 1) and if we can reach the last element, we will compare with an element which is outside the legal range of the array. That's why we go until the length-1 element.

- In the middle of the repetitive instruction we have a very interesting piece of code, namely we are swapping the values of elements if the condition is true

- You can think of this swapping procedure as we had 2 glasses with milk and we want to swap their content. Below you can see the entire algorithm for this:

    - In order to swap the content of our two glasses of milk, we need a third empty glass, of same capacity (that's why we are also choosing `temp` to have the same type as for the values that we want to interchange)
    - We empty a glass into the empty glass
    - Now, we empty the second glass into the glass that has been previously emptied
    - And last, we put the content of the spare glass into the second glass

  Note: Try to do the exercise which says that you have to swap two numbers

- The last interesting point here is that we swap the numbers if the current one is greater than the next one because we want to sort them in ascending order. If we were to need to sort them in descending order, we would just have to reverse the sign, namely to see if the current number is smaller than the next one. You also have an exercise where you can test it.

Introduction to Multidimensional Arrays

- A two-dimensional array can be declared as follows:

```
int [][] chessTable = new int[8][8];
```

    - This declares the chessTable array with 8 sets of 8 integers elements each. Note how each dimension is between its own pair of square brackets
    - You can visualize a two-dimensional array as a rectangular arrangement like, in our previous example, a table which has `8 rows` and each row has `8 columns`.

- Even though we can visualize them as rows and columns, in memory, all the elements are stored sequentially.

    - For example, let's suppos we have the following two-dimensional array: `int numbers[3][3]`
    - In memory, the elements are stored sequentially as follows:
        - `numbers[0][0] | numbers[0][1] | numbers[0][2] | numbers[1][0] | numbers[1][1] | numbers[1][2] | numbers[2][0] | numbers[2][1] |`

`numbers[2][2]`

- Another way of visualizing a two-dimensional array is like an array of elements where each element is an array itself
    - A simple array is also called a `one-dimensional` array of elements
- A 2 Dimensional array is also called a matrix

Initializing Multidimensional arrays

- If you remember from the previous lesson, a simple array can be initialized as follow:

```
double[] arr = new double[] {1.2, 2.3, 2.6, 2.8, 2.9};
```

- Now, in order to extrapolate, 2D array, can be initialized in a similar manner:

```
double[][] arr = new double[][] {
            {1.0, 2.3, 3.4},
            {1.1, 3.3, 4.4}
        };
```

    - Here, each element is an array itself, that's why we are saying that a matrix or a 2D array is an array of arrays.
- Another way of initializing a 2D array is to process all elements using a repetitive instruction (loop). In order to iterate through all elements, a nested loop is needed. See the snippet below where we initialize each element with a random number between 1 and 10 and then we display the matrix:

```
import java.util.Random;

public class Application {

    public static void main(String[] args) {
        int[][] matrix = new int[3][3];
        Random random = new Random();

        for(int i = 0; i < matrix.length;i++) {
            for(int j = 0; j < matrix[i].length; j++) {
                int randomInt = random.nextInt(11);
                matrix[i][j] = randomInt;
            }
        }

        displayMatrix(matrix);
    }

    public static void displayMatrix(int[][]matrix) {
```

```
            for(int i = 0; i < matrix.length;i++) {
                for(int j = 0; j < matrix[i].length; j++) {
                    System.out.print(matrix[i][j] + " ");
                }
                System.out.println();
            }
        }

    }
```

- Now, let's get through each of the lines from the previous JAVA snippet:
    - First, we declare a matrix which has 3 rows and 3 columns. You can also see it as an array with three elements where each element is an array which also has 3 elements
    - Next, we create an object of type `Random`. This class offers us the mechanism for generating random numbers
    - The following instructions are two nested `for` instructions. They can be seen as follows:
        - For each row
            - For each column of this row
                - Generate a random number between 0 and 11 (not including 11). That's what the `random.nextInt(11) does`
                - Set the value of the element which can be found on the current row and the current column to be equal to the newly generated random number

Note: the first `.length` tells us how many rows we have and the second `.length` tells us how many columns does the current row has

Accessing elements in a 2D array:

- Despite simple arrays, or one-dimension array, where you had to specify only one value, in order to access an element in a two-dimension array, you have to specify two values:
    - one for the row of your element
    - and another one for the column of your element
    - e.g: in order to access the element on the third row and on the 4th column, in a two-dimension array called numbers, we type: `numbers[3][4]`.

Class exercises

1. Write a JAVA program which prints the element on the main diagonal of a two-dimension array with n rows and n columns.
    - Input:

```
1 2 3
4 5 6
```

```
7 8 9
```

- Output: `The elements on the main diagonale are: 1 5 9`
- Solution:

```
public class Application {
    public static void main(String[] args) {
      int[][] matrix = new int[][] {
        {1,2,3},
        {4,5,6},
        {7,8,9}
      };

      System.out.print("The elements on the main diagonale are: ");
      for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix[i].length; j++) {
          if(i == j) {
            System.out.print(matrix[i][j] + " ");
          }
        }
      }
    }
}
```

- Explanation:
  - As you can see, the elements on the main diagonale are `1, 5, and 9`.
  - The indexes corresponding to these elements are:
    - `matrix[0][0]`
    - `matrix[1][1]`
    - `matrix[2][2]`
  - We can easily observe that the formula for a number to be on the main diagonale is `matrix[row][col]` where `row` and `col`, which represents the corresponding index of the row and column, have the same value.

2. Write a JAVA program which displays the numbers on the second diagonale of a 2D array

- Input:

```
1 2 3
4 5 6
7 8 9
```

- Output: `The elements on the second diagonale are: 3 5 7`
- Solution:

```
public class Application {

    public static void main(String[] args) {
      int[][] matrix = new int[][] {
        {1,2,3},
        {4,5,6},
        {7,8,9}
      };

      System.out.print("The elements on the second diagonale are: ");
      for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix[i].length; j++) {
          if(j == (matrix[i].length - i -1)) {
            System.out.print(matrix[i][j] + " ");
          }
        }
      }
    }
  }
```

- Explanation:
  - As we can see, the elements on the second diagonale are: `3,5, and 7`
  - Their corresponding indexes are:
    - `matrix[0][2] | matrix[1][1] | matrix[2][0]`
  - If we try to create a formula, based on the number of columns which gives us these number, we reach the following:
    - let `n` be the number of rows meaning `3`
      - for the first number we see that that we can obtain the indexes as follows
        - `3 (number of columns) - 0 - 1` for the row
        - `0` for the column
      - for the second number we can obtain the indexes by:
        - `3 (number of rows) - 1 -1` for the row
        - `1` for the column
      - for the third number we can obtain the indexes by:
        - `3 (number of rows) -2 -1` for the row
        - `2` for the column
    - We can generalize the formula as follows:
      - `n-1-i` for the row index
      - `i` for the column index

3. Write a JAVA program which reads a matrix from the standard input and computes its transpose

- Sample Input:

```
1 2 3
4 5 6
7 8 9
```

- Sample Output:

```
1 4 7
2 5 8
3 6 9
```

- Solution:

```
public class Application {

  public static void main(String[] args) {
    int[][] matrix = new int[][] {
      {1, 2, 3},
          {4, 5, 6},
          {7, 8, 9}
    };

    int[][] transpose= new int[3][3];
    for(int i =0; i < matrix.length; i++) {
      for(int j = 0; j < matrix[i].length; j++) {
        transpose[i][j] = matrix[j][i];
      }
    }

    for(int i =0; i < transpose.length; i++) {
      for(int j = 0; j < transpose[i].length; j++) {
        System.out.print(transpose[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

- Explanation:
  - By transpose of a matrix, we understand that the rows becomes columns and viceversa.
  - To start, we declare another two dimensional array which will be able to store the transpose.
  - Then we iterate through the elements of the original matrix, and at each step, we simply store at the current position (`transpose[i][j]`), the element which can be found at the location given by `j` for the rows and `i` for the columns

4. Write a JAVA program which will display the elements from a matrix, which are not present on either the diagonale (nor main, or second)
   - Sample Input:

```
12 13 21 17
8  9  15 4
2  3  7  9
21 24 29 18
```

- Sample Output:

```
13  21
8   4
2   9
24  29
```

- Solution:

```
public class Application {

  public static void main(String[] args) {
    int[][] matrix = new int[][] {
           {12, 13,  21, 17},
            {8,   9,  15,  4},
            {2,   3,  7,  9},
            {21,  24, 29, 18}
    };

    for(int i =0; i < matrix.length; i++) {
       for(int j = 0; j < matrix[i].length; j++) {
           if( i!=j && j != (4 - i -1)) {
             System.out.print(matrix[i][j] + " ");
           }
       }
       System.out.println();
    }
  }
}
```

Homework exercises

1. Create a JAVA program which reads two number from the Standard Input and then interchange them
2. Create a JAVA program which reads n values from the Standard Input and then put them inside an array. The program should sort the array in ascending order and then display the sorted array

- Sample Input:
  - n = 5
  - [5,1,4,2,8]
- Sample Output:
  - 1 2 4 5 8

3. Create a JAVA program which reads n values from the Standard Input and then put them inside an array. The program should sort the array in descending order and then display the sorted array

- Sample Input:
  - n = 5
  - [5,1,4,2,8]
- Sample Output:
  - 8 5 4 2 1

4. Write a JAVA program which computes the harmonic mean (media armonica) of the numbers on the second diagonale.
   - Input:
     ```
     1  2  3  4
     5  5  7  8
     9  10 11 12
     13 14 15 15
     ```
   - Output: `The harmonic mean of the numbers on the second diagonale is: 8,08`

5. W0rite a JAVA program which computes the sum of the numbers on the main diagonale and the sum of the numbers on the second diagonale and then displays which one is greater
   - Input:
     ```
     1  2  3  4
     5  5  7  8
     9  10 11 12
     13 14 15 15
     ```
   - Output: `The second diagonale has the sum greater.`

6. Write a JAVA program which displays if a certain number, read from the standard input, is present in a Two-Diagonale array. Also the program should display how many times the number has appeared in the matrix
   - Input: Enter the number to search for: 5
     ```
     1  2  3  4
     5  5  7  8
     9  10 11 12
     13 14 15 15
     ```
   - Output: `The number appears in the matrix 2 times`

- Input 2: Enter the number to search for: 5

```
1  2  3  4
5  5  7  8
9  10 11 12
13 14 15 15
```

- Output: `The number could be found in the matrix`

Guidelines

- As always, try to extract the classes that you encounter (and here I am speaking about the classes which are not created by you). For example, try to play with the Random class, to create new objects and generate various random numbers between various intervals