Learning goals

Before the next day, you should have achieved the following learning goals:

- Become familiarised with the use of `private` and `public`. All your classes, fields, and methods should specify explicitly wheter they are public or private according to the rules of thumb in the notes. If you make a decision of visibility that deviates from those rules, you should explain why in a comment.
- Related to the former point, you should become used to use constructors in all your classes. The constructor method or methods should be used to initialise the fields of any new object of that class.
- Be able to create classes in their own `.java` file, compile them using `javac`, and use those classes from Groovy or Java Decaf programs.
- Be able to cast simple types from one type to another.
- Be able to create and use arrays in one or more dimensions.

You should be able to finish most of non-star exercises in the lab. Remember that star exercises are more difficult. Do not try them unless the normal ones are clear to you.

1 Dividing integers

Create a Java class called `Calculator`. The class should implement the following methods, each of them printing the result on the screen.

- `add(int, int)`
- `subtract(int, int)`
- `multiply(int, int)`
- `divide(int, int)`
- `modulus(int, int)`

Note that you will will need to cast the parameters into `double` to perform exact division.
Write a small Groovy or Java Decaf program that uses all the methods of `Calculator`.

2 Checking arrays

Create a new Java class called `ArrayChecker` with two methods:

isSymetrical(int[ ):] a method that returns true if the array of `int` provided as argument is symmetrical and false otherwise.

reverse(int[ ):] a method that takes an array of `int` and returns another array of `int` of the same size and with the same numbers, but in opposite order.

Write a Groovy script that creates an object of class `ArrayChecker` and uses its methods to check whether a few arrays are symmetrical and, if they are not, reverses them.

3 Copying arrays

Create a new Java class called ArrayCopier with a method called `copy` that takes two arrays of integers as parameters. The method should copy the elements of the first array (you can call it `src`, from "source") to the second one (`dst`, from "destination") as much as possible.

If the second array is smaller, then only those elements that fit will be copied. If the second array is larger, it will be filled with zeroes.

Write a Groovy script that creates an object of class `ArrayCopier` and uses its `copy` method to copy some arrays in all three cases:

- Both arrays are of the same size.
- The source array is longer.
- The source array is shorter.

4 Creating matrices

Create a class Matrix that has a 2-D array of integers as a field. The class should have methods for:

- a constructor method `Matrix(int,int)` setting the size of the array as two integers (not necessarily the same). All elements in the matrix should be initialised to 1.
- a method `setElement(int,int,int)` to modify one element of the array, given its position (the first two integers) and the new value to be put in that position (the third integer). The method must check that the indeces are valid before modifying the array to avoid an `IndexOutOfBoundsException`. If the indeces are invalid, the method will do nothing and the third argument will be ignored.
- a method `setRow(int,String)` that modifies one whole row of the array, given its position as an integer and the list of numbers as a String like "1,2,3". The method must check that the index is valid and the numbers are correct (i.e. if the array has three columns, the String contains three numbers). If the index or the String is invalid, the method will do nothing.
- a method `setColumn(int,String)` that modifies one whole column of the array, given its position as an integer and the list of numbers as a String like "1,2,3". The method must check that the index is valid and the numbers are correct (i.e. if the array has four rows, the String contains four numbers). If the index or the String is invalid, the method will do nothing.
- a method `toString()` that returns the values in the array as a String using square brackets, commas, and semicolons, e.g. "[1,2,3;4,5,6;3,2,1]".
- A method `prettyPrint()` that prints the values of the matrix on screen in a legible format. Hint: you can use the special character '\t' (backslash-t) to mark a tabulator so that all numbers are placed in the same column regardless of their size. You can think of a tabulator character as a move-to-the-next-column command when printing on the screen.

Create a Groovy program that uses all those methods from the Matrix class: creates matrices, modifies its elements (one-by-one, by rows, and by columns), and prints the matrix on the screeen.

5 One-liners for matrices (*)

Extend your `Matrix` class with a method `setMatrix(String)` that takes a String representing the numbers to be put in the elements of the array separated by commas, separating rows by semicolons, e.g. 1,2,3;4,5,6;7,8,9.

6 Symmetry looks pretty

Make a class `MatrixChecker` with three methods:

- `isSymmetrical(int[])` takes an array of `int` and returns true if the array is symmetrical and false otherwise. An array is symetrical if the element at [0] is the same as the element at [length-1], the element at [1] is the same as the element at [length-2], etc.
- `isSymmetrical(int[][])` takes an bidimensional array of `int` and returns true if the matrix is symmetrical and false otherwise. An matrix is symmetrical if `m[i][j] == m[j][i]` for any value of `i` and `j`.
- `isTriangular(int[][])` takes an bidimensional array of `int` and returns true if the matrix is triangular[1] and false otherwise. An matrix is triangular if `m[i][j] == 0` for any value of `i` that is greater than `j`.

Add some methods to your `Matrix` class from the other exercise to perform tests on the matrices you create using the methods from MatrixChecker. (Hint: these methods will need to create objects of type `MatrixChecker`).

[1] A matrix can be up-triangular or low–triangular, but just checking one of the two is fine for this exercise.

7 Anti-aircraft aim (*)

Create an enumerated type `Result` in its own file. The `enum` must have 8 possible values: HIT, FAIL_LEFT, FAIL_RIGHT, FAIL_HIGH, FAIL_LOW, FAIL_SHORT, FAIL_LONG, OUT_OF_RANGE. Hint: the `enum` must be `public`.

Then create a Java class `Target` with the following methods:

- A constructor method `Target(int)` that creates a 3-D array of integers of the proposed size in all three dimensions. All elements must be set to zero.
- A method called `init()` that sets all the elements in the matrix to 0 except one —selected randomly— that will be set to 1. Hint: Remember that you can get a random integer between 0 and N (not including N) by using `int numberToGuess = (int) Math.abs(N * Math.random())`.
- `fire(int,int,int)` a method that checks whether the element determined by the indexes is 1 and returns a type `Result` according to the result: `Result.HIT` if the element is 1, `Result.FAIL_LEFT` if the element of value one is "to the left" (you must decide what left and right are in your 3-D array), etc. If any of the indeces is too big (or negative), the method must return `Result.OUT\_OF\_RANGE`. Left–right information takes precedence over high–low, and this takes precendence over short–long. If the 1 is to the left and behind, the output should be `Result.FAIL\_LEFT`.

Write a small program that tells the user they must hit a flying target, and then let the user try to find it by introducing three indeces. The program should use an object of class `Target` to know whether the user hit or not, and provide feedback accordingly. Here is a sample out of such a program in a space $10x10x10$.

```
Here they come! Try to bring the plane down!
Enter a coordinate X: 30
Enter a coordinate Y: 4
Enter a coordinate Z: 5
That shot is way out of range. Try harder!
Enter a coordinate X: 3
Enter a coordinate Y: 4
Enter a coordinate Z: 5
You missed! The target is to the right!
Enter a coordinate X: 5
Enter a coordinate Y: 4
Enter a coordinate Z: 1
You missed! The target is farther!
Enter a coordinate X: 5
Enter a coordinate Y: 4
Enter a coordinate Z: 5
You hit it! Well done!
Would you like to play again? y
Here they come! Try to bring the plane down!
Enter a coordinate X:
```

8 Big enough (*)

Write a small program that asks for the names and IDs of all employees in a small company, and store them in an array of integers and an array of Strings. (You will need to create a Java class to hold the arrays, and to access them).

This is similar to the example from the notes, but you do not know the number of employees in advance. Read the names and IDs of employees until the user enters an empty name (i.e. length 0) or an ID equal to 0.

Once you have finished reading employee data, go through the employee list and print the names and IDs of those employees whose ID is even or their names start with "S".

(Hint: As you do not know how many employees you need in advance, you will need a growing array. Create a small array, if it gets full create an array twice as big, copy all data to the new array, and discard the old array, etc).