Lab session 0. Revisiting Polymorphism

1. Presentation

To put in practice the concept of polymorphism.

The UML diagram class at the last page of this document shows several hierarchies of interfaces and classes for solving linear systems of equations. These hierarchies and the tasks are presented in the next sessions.

2. Hierarchy for LinearSystem

Below follows a description of the components of this hierarchy as well as the tasks that you have to perform.

1. `LinearSystem` abstract class is the superclass of all the types of linear systems existing: systems of linear equations and systems of eigenvalues. In this session you will only work with the first type, managed with objects of class `EqsLinearSystem`.
2. `EqsLinearSystem` . A system of linear equations has, in addition to its matrix, a vector, managed with objects of class `Vector`. A vector shall be supported with an unidimensional array of double values.
3. `Matrix` interface provide the methods for setting and getting values in in and from a matrix, regardless its supporting structure.
4. You have to develop two classes implementing the interface `Matrix`, namely: `FullMatrix` (a bidimensional array of double values), and `MapMatrix` (where the matrix is supported by a map whose keys are strings which indicate the row and the column, and the values are the corresponding double values in that position of the matrix).

You have to complete all the classes and interfaces present in this hierarchy following the specifications of the methods that appear in their java code.

3. Hierarchy for Solvers

The other hierarchy of classes include classes that are able of solving the linear systems in the former hierarchy using different methods.

1. `LinearSystemSolver` abstract class is the superclass for all the classes that solve a linear system, generating a solution.
2. `LinearSystemSolution` interface is an interface that any class supporting a solution of a linear system has to implement. Note that for the moment, this interface does not have any method specified. This is not so infrequent, as it only serves for allowing that a certain class declares that it also supports a solution of a linear system. Note that in our UML class diagram class `Vector` is defined as implementing this interface: a vector is a solution of a system of linear equations. But for a system of eigenvalues, one could be interested in defining the solution in terms of eigenvalues and eigenvectors.
3. `EqsLinearSystemSolver` abstract class is the superclass for all the classes that solve a linear system of equations, generating a solution as a vector.

4. Classes `GaussEqsLinearSystemSolver` and `GaussJordanEqsLinearSystemSolver` are classes that implement the Gauss elimination and the Gauss-Jordan elimination methods respectively.
5. In the link below you will find Java code for Gauss method. https://www.sanfoundry.com/java-program-gaussian-elimination-algorithm/

6. In the link below you will find Java code for Gauss-Jordan method. https://www.geeksforgeeks.org/program-for-gauss-jordan-elimination-method/

You must program both methods in the former classes. But the code in these classes must use ONLY methods of `LinearSystem`, `EqsLinearSystem`, `Matrix` and `Vector`. Note that `LinearSystem`, `EqsLinearSystem` and `Matrix` are abstractions (abstract classes or interfaces), which means that most of the code of resolution of a system of linear equations will be programmed against abstractions, which therefore means that this code will work regardless the structure that supports the matrix of the system (a bidimensional array or a map)!.

4. Running

Now in the Main class you have to generate code for doing what is indicated in comments in the main() method, namely:

1. Create the matrix indicated after this bulleted list supported by a `FullMatrix` object
2. Create the vector indicated after this bulleted list for the system of linear equations
3. Create a linear system supported by `EqsLinearSystem` whose matrix and vector are the matrix and vector previously created
4. Create a solver for solving the previous system of linear equations using gauss elimination
5. Solve the system and print the solution
6. Now create a solver for solving the previous system of linear equations using gauss-jordan elimination
7. Solve the system and print the solution
8. Repeat steps 1 to 7 but supporting the matrix of the system with an object `MapMatrix`

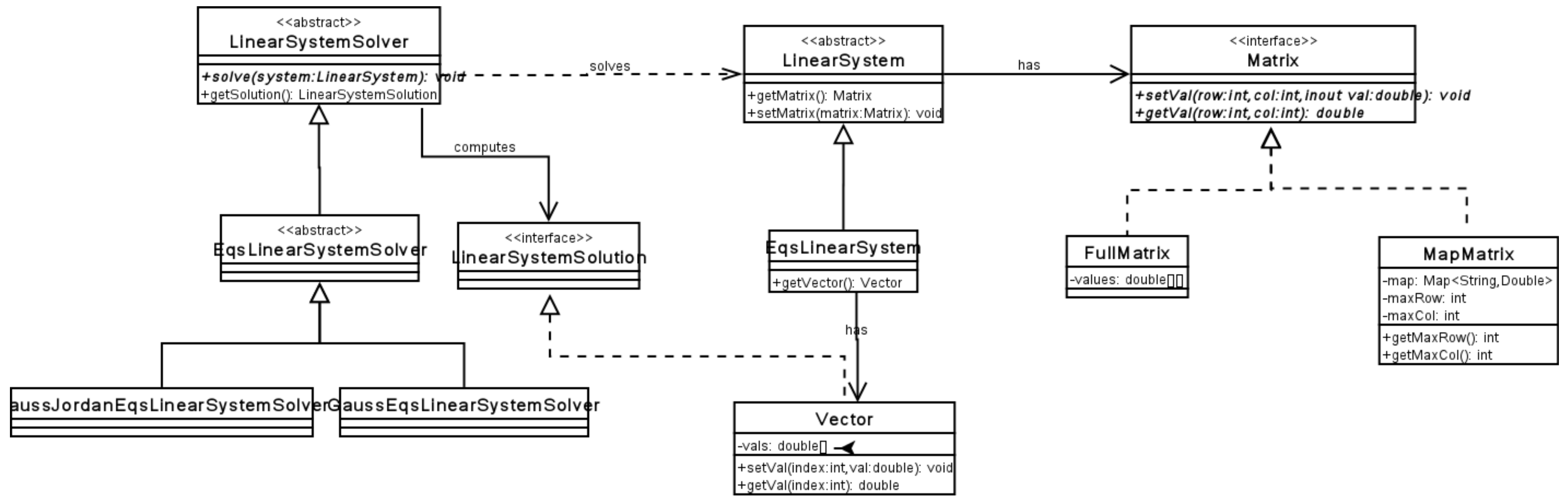
The system of linear equations to create shall be the next one:

$$\begin{bmatrix} 4 & -2 & 0 & -2 & 0 & 0 \\ -1 & 4 & -1 & 0 & -2 & 0 \\ 0 & -1 & 4 & 0 & 0 & -2 \\ -1 & 0 & 0 & 4 & -2 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} \emptyset_1 \\ \emptyset_2 \\ \emptyset_3 \\ \emptyset_4 \\ \emptyset_5 \\ \emptyset_6 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

It is generated after applying the Finite Differences numerical method for trying to solve the problem of elastic torsion of prismatic bars (example taken from "Finite Elements and Approximation" by O.C. Zienkewics and K.Morgan; John Wiley & Sons, 1983).

The solution of this system is:

$$\begin{bmatrix} 3{,}1370 \\ 2{,}8866 \\ 1{,}9971 \\ 2{,}3873 \\ 2{,}2062 \\ 1{,}5508 \end{bmatrix}$$



<<abstract>>
LinearSystemSolver
+solve(system:LinearSystem): void
+getSolution(): LinearSystemSolution
solves
<<abstract>>
LinearSystem
+getMatrix(): Matrix
+setMatrix(matrix:Matrix): void
has
<<interface>>
Matrix
+setVal(row:int,col:int,inout val:double): void
+getVal(row:int,col:int): double
computes
<<abstract>>
EqsLinearSystemSolver
<<interface>>
LinearSystemSolution
EqsLinearSystem
+getVector(): Vector
has
FullMatrix
-values: double[][]
MapMatrix
-map: Map<String,Double>
-maxRow: int
-maxCol: int
+getMaxRow(): int
+getMaxCol(): int
aussJordanEqsLinearSystemSolver
GaussEqsLinearSystemSolver
Vector
-vals: double[]
+setVal(index:int,val:double): void
+getVal(index:int): double