Lab Exercice 3

Ivo Sequeros del Rey NIA: 183711 Paula Bassagañas Odena NIA: 158710

THE PROBLEM TO SOLVE

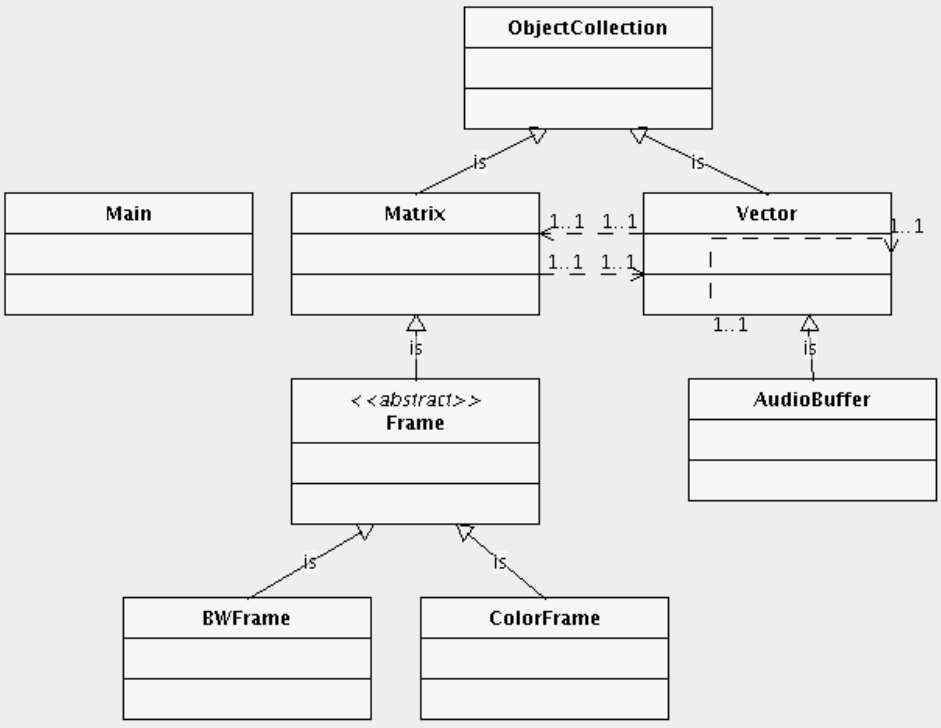
THE PROBLEM

The problem to solve is that described in the seminar session 3. The issue to solve can be summarised as follows:

- **Design of 8 classes:** ObjectCollection, Matrix (extends ObjectCollection), Vector (extends ObjectCollection), Frame (extends Matrix), BWFrame (extends Frame), ColorFrame (extends Frame), AudioBuffer (extends Vector)
- Matrix and Vector shall use a simple Array, instead of an ArrayList, to minimise load.

WRITTEN CLASSES

In order to solve the problem above we have implemented the 8 classes:



Ivo Sequeros del Rey, Paula Bassagañas Odena

METHOD & CLASS DESCRIPTIONS

After the past two lab exercises, we have concluded that it is no longer optimal to include the methods and classes descriptions on our report. Instead, we have furthered our efforts to create consistent Java docs. While writing each class and each function, we have included the following header:

```
/**
 * <Class/Method title>
 * Full description
 * @author x and y
 */
```

This allows us to be more precise in our descriptions: we also include a description for every argument and every Exception thrown by the method. The full resulting java docs are included in the zip root inside a folder called ‘docs’.

A sample of one of this docs, that of Matrix, can be found on the next page.

THE SOLUTION

We have build a library that fulfils all proposed requirements. Vector and Matrix have been the most challenging, as they needed to control a double and Vector matrix with a maximum size that had a different operational size.

We have made some manual implementation tests on the 'Main' class. They produce the following output. We have checked that all data were correct.

```
-----------------------------------------------
---------------------------------------

EXERCICI 1 : Vector constructor && set Method

1.1. Now we print a column vector:  v = [1 2 3]

OUTPUT: The vector printed after initializing
it and setting values is:

1.0

2.0

3.0

1.2. Now we print a zero column vector:  v = [0
0 0] using zero() method

OUTPUT: The vector printed after using zero()
method is:

0.0

0.0

0.0

-----------------------------------------------
---------------------------------------

EXERCICI 2: Matrix constructor && set Method

2.1. Now we print a 2x2  matrix

OUTPUT: The matrix printed after initializing
it and setting values is:

1.0 0.0

0.0 1.0

2.2. Now we print a zero matrix by applying
zero() method to the previous matrix

OUTPUT: The matrix printed after initializing
it and setting values is:

0.0 0.0

0.0 0.0

-----------------------------------------------
---------------------------------------

EXERCICI 3: Rotation matrices

3.1. Now we print a 3D rotation matrix and a
rotation vector

OUTPUT: The matrix printed after initializing
it and setting values is:

0.0 -1.0 0.0
```

1.0 0.0 0.0

0.0 0.0 1.0

1.0

0.0

0.0

3.2. Now we will rotate the vector using the 3D rotation matrix

OUTPUT: The rotated vector is:

6.123233995736766E-17

1.0

0.0

EXERCICI 4: Adding rows and columns

4.1. We will add a row and a column to the following matrix

0.0 -1.0 0.0

1.0 0.0 0.0

0.0 0.0 1.0

OUTPUT: The matrix with the added row is:

0.0 -1.0 0.0

1.0 0.0 0.0

0.0 0.0 1.0

0.0 0.0 0.0

OUTPUT: The matrix with the added column is:

0.0 -1.0 0.0 0.0

1.0 0.0 0.0 0.0

0.0 0.0 1.0 0.0

0.0 0.0 0.0 0.0

0.0 -40.0 0.0 0.0

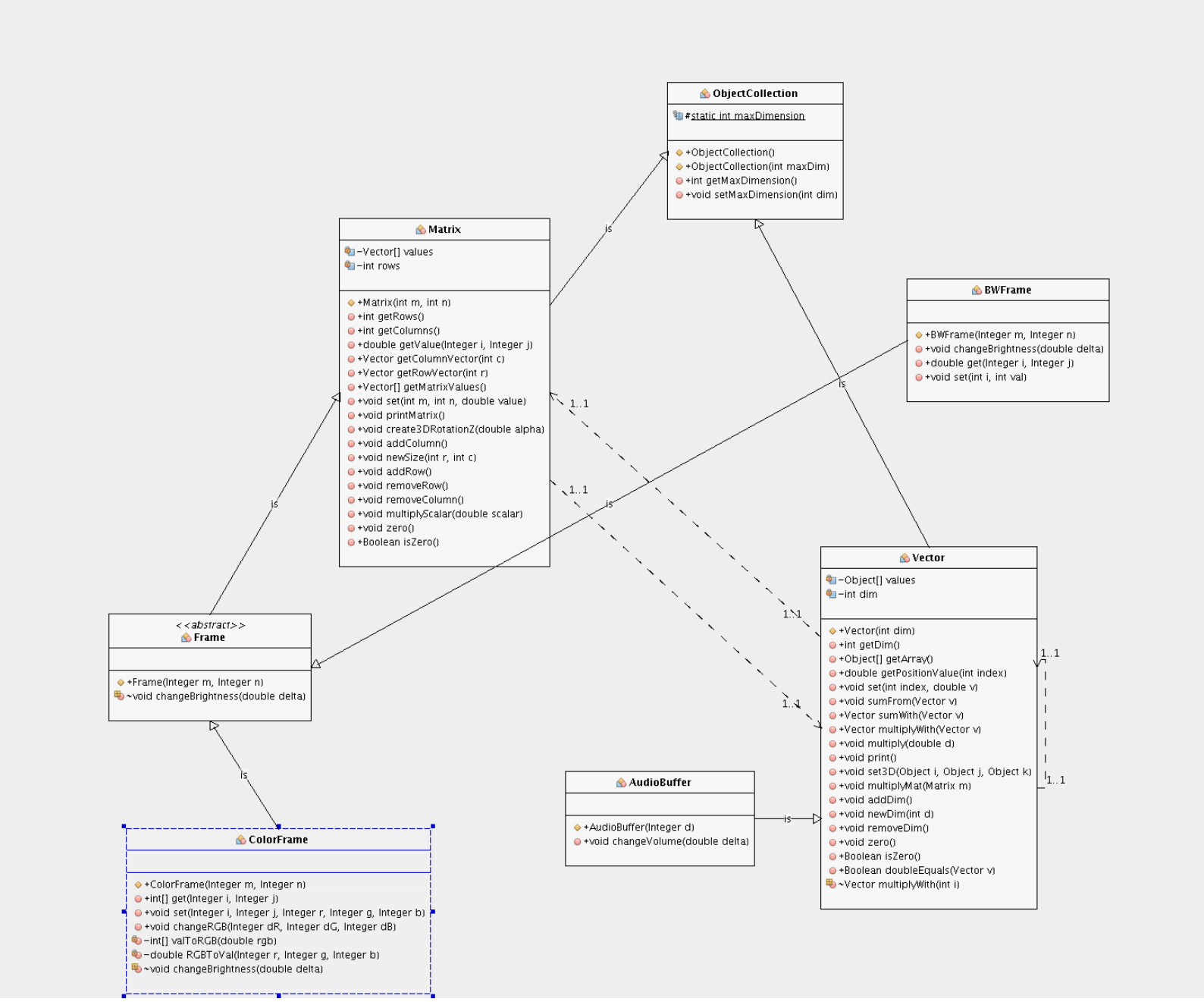
40.0 0.0 0.0 0.0

0.0 0.0 40.0 0.0

0.0 0.0 0.0 0.0

Ivo Sequeros del Rey, Paula Bassagañas Odena

Below is included a graph of all the classes, methods and attributes of this library.



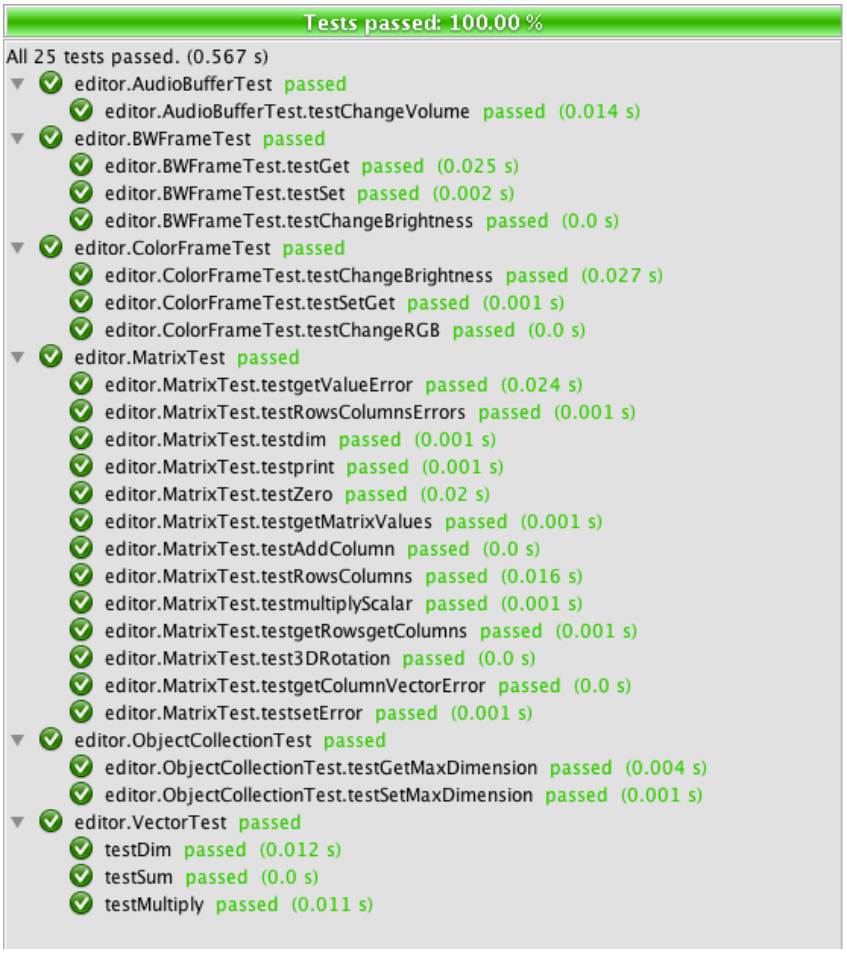
CONCLUSIONS

TESTS

We have built up a suit of unit tests UTs that ensure the correct behaviour of methods of our program. This is a common practice in big projects that are thought to be scaled in order to ensure that each small piece of code is working as expected. Once the testing infrastructure is build up it can be used to implement a gated check-in so that no user can add code to a project if that code does not pass all unit tests. Of course UTs does not ensure completely that the program is working, and that's why usually integration tests are performed together with manual tests.

In this case, we have written **25 tests**, with which we have achieved **89% coverage** of our code. We have used plugin called JaCoCo in order to see the code coverage. While implementing UTs we redesigned our Vector and Matrix classes to make sure they were fully implementable. We added removeCols, removeRows, removeDim, newSize methods to allow for Vector and Matrix downscale. Unit testing has taken more time than expected with this library, as the Exceptions implementation has made the number of instruction branches grown significantly. In our endeavour to test as much code as possible, we have also included a set of tests that check that all Exceptions are thrown when needed.

Here are the find the results of the UT:



We also include the coverage of our tests below:

editor

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Vector		76%		67%	19	48	32	107	3	19	0	1
BWFrame		71%		n/a	1	4	2	9	1	4	0	1
Matrix		100%		100%	0	45	0	83	0	18	0	1
ColorFrame		100%		n/a	0	7	0	21	0	7	0	1
ObjectCollection		100%		n/a	0	4	0	9	0	4	0	1
AudioBuffer		100%		n/a	0	2	0	4	0	2	0	1
Frame		100%		n/a	0	1	0	2	0	1	0	1
Total	136 of 1,184	89%	19 of 112	83%	20	111	34	235	4	55	0	7

As the library's most complex class, Matrix has taken a significant amount of time to be tested. We have achieved 100% coverage of this class:

Matrix

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
printMatrix()		100%		100%	0	3	0	8	0	1
newSize(int, int)		100%		100%	0	9	0	15	0	1
create3DRotationZ(double)		100%		100%	0	3	0	8	0	1
getRowVector(int)		100%		100%	0	3	0	6	0	1
isZero()		100%		100%	0	3	0	5	0	1
multiplyScalar(double)		100%		100%	0	2	0	3	0	1
Matrix(int, int)		100%		100%	0	2	0	7	0	1
removeRow()		100%		100%	0	2	0	3	0	1
zero()		100%		100%	0	2	0	3	0	1
getValue(Integer, Integer)		100%		100%	0	3	0	4	0	1
addColumn()		100%		100%	0	2	0	3	0	1
addRow()		100%		100%	0	2	0	3	0	1
set(int, int, double)		100%		100%	0	3	0	6	0	1
removeColumn()		100%		n/a	0	1	0	3	0	1
getColumnVector(int)		100%		100%	0	2	0	3	0	1
getRows()		100%		n/a	0	1	0	1	0	1
getColumns()		100%		n/a	0	1	0	1	0	1
getMatrixValues()		100%		n/a	0	1	0	1	0	1
Total	0 of 501	100%	0 of 54	100%	0	45	0	83	0	18

We have also achieved 100% coverage on all branches of the following classes:

ColorFrame

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
valToRGB(double)		100%		n/a	0	1	0	5	0	1
RGBToVal(Integer, Integer, Integer)		100%		n/a	0	1	0	2	0	1
set(Integer, Integer, Integer, Integer, Integer)		100%		n/a	0	1	0	3	0	1
get(Integer, Integer)		100%		n/a	0	1	0	4	0	1
changeRGB(Integer, Integer, Integer)		100%		n/a	0	1	0	3	0	1
ColorFrame(Integer, Integer)		100%		n/a	0	1	0	2	0	1
changeBrightness(double)		100%		n/a	0	1	0	2	0	1
Total	0 of 92	100%	0 of 0	n/a	0	7	0	21	0	7

AudioBuffer

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
AudioBuffer(Integer)		100%		n/a	0	1	0	2	0	1
changeVolume(double)		100%		n/a	0	1	0	2	0	1
Total	0 of 9	100%	0 of 0	n/a	0	2	0	4	0	2

Frame

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
Frame(Integer, Integer)		100%		n/a	0	1	0	2	0	1
Total	0 of 7	100%	0 of 0	n/a	0	1	0	2	0	1