7. homework assignment; JAVA, Academic year 2013/2014; FER

A usual, please see the last page. I mean it! You are back? OK. This homework consists of one problem.

Problem 1.

As part of this problem you will develop a simple library for linear algebra. You should place all of developed classes into package `hr.fer.zemris.linearna`.

I have put in the repository on Ferko the following Java archive file: `lin-models.jar`. Download it and include it in your project (`lib` folder). This JAR has the definitions of two important interfaces (`IVector`, `IMatrix`) and two exceptions (`IncompatibleOperandException`, `UnmodifiableObjectException`). In order to compile you program using *ant* make sure that you have `lib` folder added in appropriate *classpath*.

An example of not to complicated library for vectors and matrices is described in first laboratory exercise at Interactive Computer Graphics course which is available in following document:

http://java.zemris.fer.hr/nastava/irg/labosi-0.1.2013-03-15.pdf

Please read carefully the library's description. For all classes a class diagram is given illustrating the relationships among various classes. File `lin-models.jar` contains both the source code with appropriate documentation and the bytecode.

Your task is to create an appropriate implementation of given interfaces and described classes. In case you have additional questions, you can google, consult the official course book:

http://java.zemris.fer.hr/nastava/irg/knjiga-0.1.2014-02-07.pdf

or come to consultations.

Once done, please equip your build.xml with target "jar" which will generate `lin-impl.jar` and `lin-demos.jar`. File `lin-impl.jar` must include all classes comprising the library implementation (but without the files already included in `lin-models.jar`) and without the subpackage `demo` defined later. File `lin-demos.jar` must include bytecode for `demo` subpackage. In your project, add a subpackage `demo` (reminder: it must be excluded when generating `lin-models.jar`), and add in it the following programs (by program I mean a class with `main` method):

- Prog1: represents implementation of example 1.1.2,
- Prog2: represents implementation of example 1.2.1,
- Prog3: represents implementation of example 1.2.2,
- Prog4: represents implementation of example 1.2.3,
- Prog5: represents implementation of example 1.2.4.

You don't have to enable user to run any of these programs directly from ant. However, once the user completes:

```
ant compile
```

writing the command:

```
java -cp lib/lin-models.jar:dist/lib/lin-impl.jar:dist/lib/lin-demos.jar
     hr.fer.zemris.linearna.demo.Prog1
```

should run selected example (the previous command should be considered as a single line).

For later homework you will need this library. Then you will simply add include `lin-models.jar` and `lin-impl.jar` into project's `lib` directory and use it.

Note. When user attempts to invert a matrix which is singular, an appropriate exception should be thrown. The performance of implemented inversion algorithm is not crucial: you don't have to code some state-of-the-art algorithm – just find some which works reasonably fast. In javadoc comment for this method add an URL which will direct the user to a site explaining selected algorithm.

Note 2. Whan of the reasons for implementing `AbstractMatrix` is to implement all of the code which is common for all implementations into a single class. This class, however, must be abstract since it is not known how to create another instance of the matrix which has the same implementation as the current one (method `newInstance()`). Once you start creating final implementations (i.e. a matrix which holds all of it data in `double[][]`), you will be able to provide a concrete implementation for methods like the mention-one. For classes which are views, delegate this decision to the observed matrix. For cases where this is not possible (e.g. you are looking at matrix as a vector), delegate this decision to factory methods of class `LinAlgDefaults`. You can always assume that `newInstance()` will create modifiable instances. The same also holds to vectors.

Class `LinAlgDefaults` is responsible for creating default implementations of vectors and matrices. It is not described in laboratory exercise so here I will give a short description. Create this class and define two public static methods:

```
public static IMatrix defaultMatrix(int rows, int cols);
public static IVector defaultVector(int dimension);
```

Implement `defaultMatrix` to create a new instance of modifiable `hr.fer.zemris.linearna.Matrix` and `defaultVector` to create a new instance of modifiable `hr.fer.zemris.linearna.Vector`. If you later add several different implementations for vectors and matrices (e.g. for working with sparse matrices), these methods can be used by inexperienced user to pick a default implementation.

You must unit-test your implementation of vector. You are encouraged to also unit-test other classes.

Important notes

Solve all of the problems in a single Eclipse project. Configure Eclipse to use two source directories: `src/main/java` for your source files and `src/test/java` for sources files of unit tests.

You are required to write the adequate number of unit tests for vectors.

You must equip your project with `build.xml` script so that the project can be build from the command line. In that script, you must integrate all of the quality-checking tools which I have described in the book (by integrate, I mean have a new target for each tool so that you can call each tool separately). It should be possible to run at least the following targets: `init`, `compile`, `compile-tests`, `jar`, `run-tests`, `quality`, `reports`, `clean`.

Target `quality` must run unit tests and all of the quality checks (i.e. *checkstyle*, *pmd*, *findbugs*): make it just a target which depends on all tool-targets. Unit tests must be run with the code coverage analysis. Target `reports` is a wrapper that will run all of the unit tests, the quality checks and the javadoc generation.

All of the classes should have appropriate javadoc.

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries which is not part of Java standard edition (Java SE) unless explicitly provided by me. This means that from this point on, you can use Java Collection Framework classes or its derivatives (moreover, I recommend it). Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. You must name your project's main directory (which is usually also the project name) `HW07-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW07-0012345678`. Once you are done, export the project as a ZIP archive and upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is April 28th 2014. at 8 AM (morning, not evening!).